INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine
Universität
Düsseldorf

Thematische Klassifikation von Partizipationsverfahren mit Active Learning

Boris Thome

Masterarbeit

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

[redacted]

[redacted]

Zusammenfassung

In dieser Masterarbeit werden verschiedene Active Learning-Verfahren im Rahmen einer thematischen Klassifikation von Texten miteinander verglichen. Active Learning ist ein Konzept aus dem Bereich des Machine Learning, das darauf abzielt die Menge der Trainingsdaten zu reduzieren. Dabei soll trotz der Verringerung der Trainingsdaten weiterhin eine möglichst hohe Güte der Ergebnisse erzielt werden. Dies wird dadurch erreicht, dass das Active Learning-Modell Anfragen an den Annotator stellen kann. Mithilfe von verschiedenen Strategien werden möglichst informative Datenpunkte für die Annotation und das darauf folgende Training gewählt. In dieser wissenschaftlichen Arbeit werden diese Strategien in Kombination mit verschiedenen Klassifikationsalgorithmen gegenübergestellt und evaluiert. Das Ziel ist es, ein möglichst effizientes Training und somit eine deutliche Reduktion des Annotationsaufwandes zu erreichen. Dies ist grundsätzlich wünschenswert, da die Erstellung einer Annotation für einen Datensatz oft sehr zeitintensiv und demnach äußerst kostspielig sein kann.
Als Beispiel lassen sich hier Partizipationsverfahren nennen, die oft von Städten oder Kreisen mit finanziell eingeschränkten Budgets durchgeführt werden. Partizipationsverfahren dienen dazu, ein besseres Verständnis für das Meinungsbild der Bevölkerung zu erhalten, indem man den Bürgern eine Plattform zur Meinungsäußerung bietet. Die daraus resultierenden Daten werden anschließend gesammelt, sodass eine Auswertung erfolgen kann. In vielen Fällen werden diese Daten jedoch aufgrund von fehlenden finanziellen Mitteln manuell ausgewertet.
Der *Raddialog* der Städte Bonn, Moers und Ehrenfeld stellt dabei einen konkreten Anwendungsfall dar. Dabei handelt es sich um ein Online-Partizipationsverfahren aus dem Jahr 2017, bei dem sich die Bürger zur Radverkehrssituation der jeweiligen Städte äußern konnten. Der daraus resultierende Datensatz besteht aus 3139 Textbeiträgen, die von einem Annotator mit einer Annotation versehen wurden. Diese Annotation beinhaltet thematische Kategorisierungen jeden Beitrages und soll hier als Grundlage für das Training eines Machine Learning-Modells unter Anwendung eines Active Learning-Verfahrens verwendet werden.

Inhaltsverzeichnis

1 Einleitung

Um eine thematische Klassifikation von Texten durchführen zu können, ist die Erstellung einer Annotation erforderlich. Eine Annotation kann als Anmerkung zu einem Textdokument betrachtet werden, die zusätzliche Informationen zu diesem Dokument enthält. So kann eine Annotation beispielsweise eine thematische Kategorisierung des Dokumentes beinhalten, die händisch erstellt wurde. Die Dokumente und deren zugehörige Annotationen können als Grundlage für das Training eines Machine Learning-Modells verwendet werden. Zunächst muss für die Annotation ein Annotationsmodell erstellt werden, in dem die Anzahl und Merkmale der jeweiligen Klassen festgelegt werden. Anschließend können die einzelnen Dokumente des Textkorpus von einem oder mehreren Annotatoren untersucht und den jeweiligen Klassen zugeordnet werden.

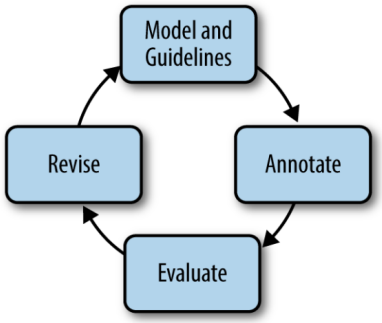

Abbildung 1: Annotationsprozess [18]

Die Annotation ist oft ein iterativer Prozess (siehe Abbildung 1), bei dem zunächst ein Modell mit Richtlinien erstellt wird. Mithilfe der definierten Regeln wird dann von einem oder mehreren Annotatoren eine erste Annotation erstellt. Optimalerweise erfolgt eine Evaluation der Annotation und eine Überarbeitung der Richtlinien. Auf Basis des überarbeiteten Modells kann der Annotationskreislauf dann erneut beginnen bis es eine ausreichende Übereinstimmung der Ergebnisse gibt. Ein Problem, das sich daraus ergibt, ist der hohe Aufwand für diesen Prozess. Um eine gute Qualität der Annotation gewährleisten zu können, ist oft der Einsatz von Spezialisten des jeweiligen Fachgebietes notwendig. Es kommt hinzu, dass Machine Learning-Algorithmen in der Regel bessere Ergebnisse erzielen, wenn ihnen eine große Menge an annotierten Trainingsdaten zur Verfügung gestellt wird. Aufgrunddessen ist für den Annotationsprozess oft ein sehr hoher finanzieller und zeitlicher Aufwand erforderlich.
Active Learning ist ein spezielles Machine Learning-Verfahren, das darauf abzielt die benötigte Menge der annotierten Daten zu reduzieren. Dabei soll die Güte der Modellvorhersage möglichst maximiert werden. Die Besonderheit von Active Learning-Verfahren besteht darin, dass der Machine Learning-Algorithmus während des Trainings Anfragen an den Annotator stellen kann. Diese Anfragen beinhalten Dokumente, die der Annotator manuell annotieren soll. Durch die gezielte Auswahl von wenigen aber dafür besonders informativen Dokumenten mithilfe von *Query Strategien* soll ein effizientes Training gewährleistet werden. Nach jeder Anfrage des Active Learning-Algorithmus

wird das Modell erneut trainiert. Zu Beginn wird eine sogenannte Abbruchbedingung definiert. Ist diese Bedingung erfüllt, so erfolgt keine weitere Iteration mehr und das Active Learning-Verfahren wird beendet.

Im Bereich der Soziologie könnte der Einsatz von Active Learning-Verfahren neue Türen für die Informationsgewinnung bei Partizipationsverfahren öffnen. Partizipationsverfahren dienen dazu, Individuen in Entscheidungs- und Willenbildungsprozesse miteinzubeziehen. Viele Städte und Kreise führen daher auf regionaler Ebene Bürgerbeteiligungsverfahren durch, bei denen beispielsweise Themen der Stadt- und Bauplanung kritisch diskutiert werden können. Ein positiver Effekt ist die direkte Integration der Bürger in den Entscheidungsprozess, sodass eine bessere Grundlage für die anstehende Entscheidungsfrage gewonnen werden kann. Aufgrund der Tatsache, dass die Teilnehmeranzahl bei Bürgerbeteiligungsverfahren oft sehr hoch ist, werden die Ergebnisse des Verfahrens meist für eine spätere Auswertung zu einem Datensatz aggregiert. Die Auswertung dieser Daten kann sehr kosten- und zeitintensiv sein und wird in vielen Fällen manuell von den zuständigen Behörden durchgeführt.
Purpura et al. [17] wandten bereits im Jahr 2008 Active Learning-Verfahren an, um eine thematische Klassifikation von Texten eines Online-Partizipationsverfahrens durchzuführen. Zum Zwecke der Klassifikation verwendeten sie *Support Vector Machines*, *Naive Bayes* und einen *Maximum Entropy Classifier*. Seitdem gab es jedoch viele Neuerungen im Bereich des Machine Learning. Mit der Veröffentlichung von modernen Machine Learning-Modellen wie *BERT* [6] und *GPT-2* [20] konnten für viele Aufgaben im Bereich des *Natural Language Processing* neue Maßstäbe gesetzt werden. Dies führte dazu, dass diese Modelle in den letzten Jahren immer öfter in Active Learning-Szenarien eingesetzt und untersucht wurden. Romberg und Escher [21] implementierten daher im Jahr 2022 ein Active Learning-Verfahren, um eine thematische Klassifikation von deutschsprachigen Texten eines Online-Partizipationsverfahrens durchzuführen. Die dafür verwendeten Daten stammen aus dem sogenannten *Raddialog*, bei dem Verkehrsthemen diskutiert wurden. Dafür wurden die Beiträge von Bürgern auf einer Online-Plattform gesammelt, in mehrere Klassen eingeteilt und annotiert. Die Beiträge wurden auf zwei verschiedene Arten in mehrere Klassen unterteilt. Die erste Unterteilung besteht aus acht Klassen, während die zweite Unterteilung aus 30 Klassen besteht. Romberg und Escher konzentrierten sich dabei zunächst auf die Klassifikation der acht Klassen und konnten in diesem Szenario vielversprechende Ergebnisse vorweisen.
In dieser Arbeit werden auf Basis des Raddialog-Datensatzes verschiedene Active Learning-Verfahren implementiert. Dabei wird der wesentliche Fokus auf die Auswahl geeigneter Klassifikationsalgorithmen und Query Strategien gelegt. Es wird sowohl die Klasseneinteilung in acht Klassen als auch die Einteilung in 30 Klassen thematisch klassifiziert und untersucht.

2 Active Learning-Verfahren

Active Learning-Algorithmen lassen sich in vier wesentliche Bausteine unterteilen: *Initialisierungsstrategie*, *Klassifikationsalgorithmus*, *Query Strategie* und *Abbruchkriterium* (siehe Abbildung 2). Diese vier Elemente lassen sich individuell austauschen, sodass viele Kombinationen dieser Bausteine möglich sind.

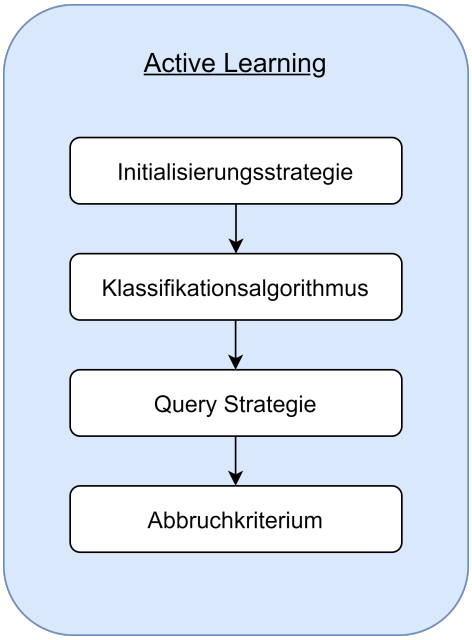

Abbildung 2: Active Learning-Verfahren Übersicht [23]

Im ersten Schritt findet eine Initialisierung statt. Dafür wird die sogenannte *Initialisierungsstrategie* benötigt. Die Initialisierungsstrategie dient lediglich dazu, eine erste Auswahl von Trainingsdaten festzulegen. Diese Auswahl von Trainingsdaten wird dann benutzt, um einen vorher definierten Klassifikationsalgorithmus, den sogenannten *Classifier*, ein erstes Mal zu trainieren. Die *Query Strategie* dient dazu, besonders informative Dokumente aus dem noch nicht annotierten Textkorpus zu extrahieren. Diese Strategien zielen darauf ab, eine möglichst gute Auswahl an Dokumenten zu treffen, sodass eine markante Verbesserung der Klassifikation trotz geringer Trainingsdatenmenge ermöglicht wird. Die ausgewählten Dokumente werden dann an einen Annotator weitergegeben, der sie annotieren soll. Nachdem die Annotation erfolgt ist, wird der Klassifikationsalgorithmus auf Basis dieser Annotation erneut trainiert. Der letzte Baustein ist das *Abbruchkriterium*. Das Abbruchkriterium ist eine Bedingung, unter der der Active Learning-Prozess gestoppt werden soll. Die Festlegung einer maximalen Anzahl an Iterationen für den Active Learning-Prozess kann somit als Abbruchkriterium betrachtet werden.

2.1 Initialisierungsstrategien

Es existieren mehrere Initialisierungsstrategien, die eine initiale Auswahl der Trainingsdaten ermöglichen. Die einfache *Random Initialization* wählt zum Beispiel eine festgesetzte Anzahl von Dokumenten zufällig aus. Da die Auswahl rein zufallsbasiert ist, ist es möglich, dass nicht alle Klassen im Trainingsdatensatz vorhanden sind. Dies hätte zur Folge, dass der Klassifikationsalgorithmus diese meist eher schwächer repräsentierten Klassen nach dem initialen Training nicht erkennen kann. Somit wäre die Güte des Modells bereits nach dem initialen Training mit hoher Wahrscheinlichkeit suboptimal, da nicht alle Klassen erkannt werden können. Liegt eine große Menge an Klassen vor, so ist es umso wahrscheinlicher, dass bei der Random Initialization nicht jede Klasse im Trainingsdatensatz repräsentiert wird. Hinzu kommt, dass die Klassenverteilung bei einem Random Sampling stark von der tatsächlichen Klassenverteilung abweichen kann. Angenommen die Klassenverteilung des gesamten Trainingsdatensatzes entspricht in etwa der Verteilung der Klassen im Gesamtdatensatz. So könnte es bei Random Sampling vorkommen, dass übermäßig viele Repräsentanten der am schwächsten vertretenen Klasse gezogen werden. Der Klassifikationsalgorithmus könnte diese Klasse dann zwar gut klassifizieren, jedoch hätte er starke Probleme bei der Klassifikation der stärker vertretenen Klassen, da er nicht genug Trainingsbeispiele für diese Klasse hatte. Dieses Problem könnte insbesondere bei einer sehr klein gewählten initialen Trainingsdatenmenge auftreten.
Es gibt verschiedene Ansätze, um eine vollständigere Repräsentation der Klassen im initialen Training zu ermöglichen. So könnte zum Beispiel beim Random Sampling einfach die Anzahl der zufällig gewählten Dokumente erhöht werden. Dies kann insbesondere bei stark imbalancierten Datensätzen dazu führen, dass die Trainingsmenge schon initial sehr hoch angesetzt werden muss, um eine vollständige Repräsentation der Klassen zu erreichen. Allerdings wird dadurch das Ziel des Active Learnings verfehlt, ein Training mit einer geringen Menge von Daten durchzuführen.
In solchen Fällen ist eine *Stratified Random Initialization* besser geeignet. Bei der Stratified Random Initialization wird der Trainingsdatensatz zunächst in mehrere sogenannte *Strata* unterteilt. Dabei entspricht ein Stratum hier jeweils einer Klasse. Innerhalb jedes Stratums wird dann ein *Random Sampling* durchgeführt, also eine einfache Zufallsauswahl an Datenpunkten. Es folgt eine *Proportional Allocation* der Strata, die als

$$\frac{n_k}{n} = \frac{N_k}{N} = w_k$$

definiert ist [7]. Dabei entspricht N der Gesamtanzahl der Daten und N_k der Gesamtanzahl der Daten pro Stratum k. Analog dazu ist n die Anzahl der auszuwählenden *Samples* und n_k die Anzahl der auszuwählenden Samples pro Stratum k. Das Gewicht, oder *Weight*, des jeweiligen Stratums ist als w_k definiert. Also entspricht die prozentuale Klassenverteilung im Gesamtdatensatz der prozentualen Klassenverteilung im Sampling.

2.2 Klassifikationsalgorithmus

Classifier sind sogenannte Klassifikationsalgorithmen und werden zur Erkennung verschiedener Klassen eines Datensatzes verwendet. Sie können auf verschiedene Art und

Weise implementiert werden. So gibt es zum Beispiel regelbasierte, statistische, überwachte und unüberwachte Klassifikationsalgorithmen. In dieser Arbeit wird der Fokus primär auf *Transformer* [28] basierte Modelle wie *BERT* [6] und *GPT-2* [20] gelegt. Diese Modelle konnten in den letzten Jahren State-Of-The-Art Ergebnisse in vielen *Natural Language Processing* Aufgaben erzielen. Da Klassifikationsalgorithmen bereits in Kapitel 3 detailliert beschrieben werden, wird an dieser Stelle nicht zusätzlich auf diese eingegangen.

2.3 Query Strategien

Query Strategien führen eine gezielte und strategische Auswahl von Datenpunkten aus dem Datensatz durch. Sie können in einem *Pool Based Active Learning-Setting* [12] eingesetzt werden, um ein effizienteres Training zu ermöglichen.

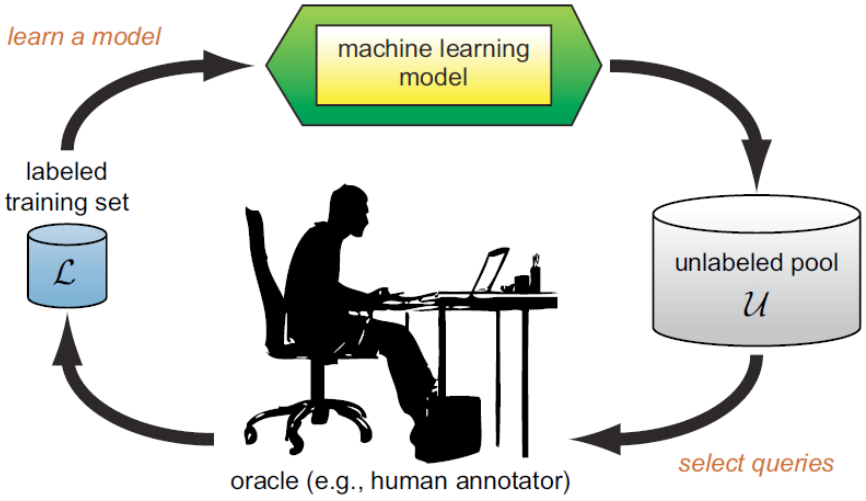

Abbildung 3: Pool-Based Active Learning [25]

Pool Based Active Learning ist ein zyklischer Prozess, bei dem wiederholt Datenpunkte aus einem großen *Pool* $\mathcal{U}$ entnommen werden (siehe Abbildung 3). Die Daten in $\mathcal{U}$ sind dabei noch nicht zuvor annotiert worden. Mithilfe einer Query Strategie werden dann wenige Datenpunkte gezielt ausgewählt und an den Annotator (oder auch *Oracle*) weitergegeben. Hat der Annotator die ausgewählten Daten mit Labels versehen, so werden die Datenpunkte in einen Trainingsdatensatz $\mathcal{L}$ aufgenommen. Anschließend wird der Klassifikationsalgorithmus mithilfe der neuen Informationen trainiert und der Kreislauf wiederholt sich. Der Active Learning-Prozess wird dann durch das vorher definierte Abbruchkriterium abgeschlossen.
Im Folgenden sollen die Query Strategien *Breaking Ties*, *Prediction Entropy* und *Contrastive Active Learning* erläutert werden. Query Strategien lassen sich in verschiedene Arten unterteilen. Query Strategien wie Breaking Ties und Prediction Entropy fokussieren sich primär auf eine Maximierung der *Confidence* des Machine Learning-Modells beziehungs-

weise eine Minimierung der *Uncertainty* eines Modells. Ist der Klassifikationsalgorithmus sich bei der Klassifikation sehr sicher, so hat er eine sehr hohe Confidence beziehungsweise eine sehr geringe Uncertainty. Die Grundidee ist also, dass eine hohe Confidence des Modells mit einer hohen Güte bei der Klassifikation einhergeht. Um also eine Erhöhung der Confidence zu erreichen, werden Datenpunkte für das Training ausgewählt, bei deren Klassifikation sich das Modell unsicher ist. Daher können solche Verfahren als *Uncertainty Based Strategies* bezeichnet werden.
Andere Query Strategien basieren wiederum auf einem *Diversity Sampling*, also einer Auswahl von möglichst heterogenen Datenpunkten. Beim Contrastive Active Learning findet eine Kombination aus *Uncertainty Sampling* und *Diversity Sampling* statt. Die Kombination dieser beiden Verfahren soll den Schwachstellen des jeweils anderen Verfahrens entgegenwirken. So kann es beim reinen Uncertainty Sampling passieren, dass trotz hoher Uncertainty Datenpunkte mit geringem Informationsgehalt gewählt werden. Das Diversity Sampling hingegen wählt zwar Datenpunkte mit hoher Heterogenität, jedoch kann es sein, dass die Datenpunkte sehr einfach zu klassifizieren sind.

2.3.1 Breaking Ties

Im Jahr 2005 präsentierten Luo et al. die Query Strategie *Breaking Ties* [13]. Diese Strategie zielt darauf ab, die *Confidence* des Machine Learning-Modells bei einer Klassifikation zu steigern. Viele Klassifikationsalgorithmen weisen einem Datenpunkt bei einer Klassifikation mehrere Wahrscheinlichkeiten zu. Diese Wahrscheinlichkeiten werden pro Datenpunkt für jede Klasse berechnet. Sie geben an, wie wahrscheinlich es ist, dass der Datenpunkt einer gewissen Klasse zugehörig ist. Das Ergebnis der Klassifikation von Datenpunkt x ist dann die Klasse p mit der höchsten Wahrscheinlichkeit $argmax_p P(p)$. Wenn nun beispielsweise die Wahrscheinlichkeit $P_x(a)$ von Klasse a für Datenpunkt x signifikant höher ist, als die Wahrscheinlichkeiten der anderen Klassen für x, so ist sich der Klassifikationsalgorithmus bei der Klassifikations sehr sicher. Er hat also eine hohe Confidence bezüglich der Klassifikation.
Bei der Durchführung von Breaking Ties werden zunächst mithilfe einer Initialisierungsstrategie Daten aus dem Pool $\mathcal{U}$ entnommen und der Klassifikationsalgorithmus damit trainiert. Daraufhin werden die Wahrscheinlichkeiten für die Klassen für alle Datenpunkte in $\mathcal{U}$ berechnet. Angenommen die Klasse mit der höchsten Wahrscheinlichkeit für Datenpunkt x ist Klasse a und die Klasse mit der zweithöchsten Wahrscheinlichkeit für x ist Klasse b. Bei der Breaking Ties Strategie wird nun die Differenz der beiden höchsten Wahrscheinlichkeiten $P(a) - P(b)$ für jeden Datenpunkt berechnet. Anschließend werden die Datenpunkte mit der geringsten Differenz $P(a) - P(b)$ in den Trainingsdatensatz $\mathcal{L}$ übernommen und an den Annotator übergeben. Es werden also diejenigen Datenpunkte in $\mathcal{L}$ übernommen, bei denen die Klassifikation äußerst knapp entschieden wurde. Dies soll insbesondere bei schwer zu klassifizierenden Datenpunkten Abhilfe schaffen, sodass ein besseres Verständnis für die Unterschiede der Klassen entsteht. Dabei ist es möglich eine Zahl n zu definieren, sodass die n Datenpunkte mit der höchsten Differenz $P(a) - P(b)$ in die Query übernommen werden.

2.3.2 Prediction Entropy

Die *Prediction Entropy* ist eine Query Strategie, die mithilfe eines geschätzten Wertes für die Entropie versucht Datenpunkte mit möglichst hohem Informationsgehalt aus dem Pool $\mathcal{U}$ zu selektieren [11]. Die Entropie gilt als ein Maß für die Uncertainty, also die Unsicherheit, eines Modells. Demnach soll die Entropie bei der Ausführung dieser Query Strategie minimiert werden, um einen maximalen Informationsgewinn zu erhalten. Holub et al. beschreiben dies zunächst als eine Maximierung der Differenz der Entropien aus Iteration t und Iteration $t+1$. Diese Differenz ist als

$$x^{(t)} = \text{argmax}_x \, \mathcal{H}\left(\mathcal{U}^{(t)}|\mathcal{L}^{(t)}\right) - \mathcal{H}\left(\mathcal{U}^{(t+1)}|\mathcal{L}^{(t+1)}\right)$$

definiert [11]. Es sei x ein Datenpunkt, t eine Iteration im Active Learning-Prozess und $\mathcal{H}$ die Entropie. Der *Unlabeled Pool* in Runde t ist $\mathcal{U}^{(t)}$ und der *Labeled Pool* in Runde t ist analog dazu $\mathcal{L}^{(t)}$. Da x hier der Datenpunkt ist, der in Runde t mit einem Label versehen wird, ist nur die zweite Hälfte des Terms von x anhängig. Somit kann die erste Hälfte $\mathcal{H}(\mathcal{U}^{(t)}|\mathcal{L}^{(t)})$ weg gelassen werden. Es ergibt sich eine Maximierung des negativen Terms $-\mathcal{H}\left(\mathcal{U}^{(t+1)}|\mathcal{L}^{(t+1)}\right)$, der wiederum zur folgenden Minimierung

$$x^{(t)} = \text{argmin}_x \, \mathcal{H}\left(\mathcal{U}^{(t+1)}|\mathcal{L}^{(t+1)}\right)$$

umformuliert werden kann [11]. Die Berechnung basiert auf $\mathcal{L}^{(t+1)}$, also der vom Orakel erhaltenen Label Information der nächsten Iteration. Dieses Wissen ist jedoch in Iteration t noch nicht vorhanden, sodass stattdessen eine Entropie in Abhängigkeit von jeder möglichen Annotation des Orakels für die Folgerunde $t+1$ berechnet wird. Die *Expected Entropy* (erwartete Entropie) ergibt sich als gewichteter Durchschnitt der Entropien. Dabei erfolgt eine Gewichtung anhand der Wahrscheinlichkeit, dass x ein bestimmtes Label vom Orakel zugewiesen bekommt. Holub et al. haben die Expected Entropy als

$$\overline{\mathcal{H}} = \sum_{j=1}^{L} P\left(Y = j|\mathcal{L}^{(t)}\right) \cdot \mathcal{H}\left(\mathcal{U}^{(t+1)}|\mathcal{L}^{(t)} \cup \{x, j\}\right)$$

definiert [11]. Es seien L die möglichen Klassen und Y eine Zufallsvariable für die Repräsentation des Labels für Datenpunkt x. Die *Minimum Expected Entropy* wählt nun also den Datenpunkt

$$x^{(t)} = \text{argmin}_x \, \overline{\mathcal{H}}$$

mit dem minimalen Wert für die Expected Entropy $\overline{\mathcal{H}}$ [11]. Um $\mathcal{H}\left(\mathcal{U}|\mathcal{L}\right)$ zu berechnen, machen Holub et al. von der *Sub-Additivity* Eigenschaft der Entropie Gebrauch. Es gilt $\mathcal{H}\left(Y_1, Y_2, \ldots, Y_M|\mathcal{L}\right) \leq \sum_{k=1}^{M} \mathcal{H}\left(Y_k|\mathcal{L}\right)$. Dabei ist M die Anzahl der verfügbaren, noch nicht annotierten Datenpunkte. Mithilfe dieser Ungleichung kann nun statt der *Joint Entropy* (kombinierte Entropie) die Summe aller individuellen Entropien berechnet werden. Diese Summe dient als obere Schranke für die Joint Entropy. Zur Berechnung von $\mathcal{H}\left(\mathcal{U}|\mathcal{L}\right)$ wird also nur die Wahrscheinlichkeitsverteilung der möglichen Labels von Y_k benötigt. Die Entropie ergibt sich daraus wie folgt:

$$\mathcal{H}\left(Y_k|\mathcal{L}\right) = -\sum_{l=1}^{L} P\left(Y_k = l|\mathcal{L}\right) \cdot \log_2 P\left(Y_k = l|\mathcal{L}\right)$$

2.3.3 Contrastive Active Learning

Contrastive Active Learning ist eine hybride Query Strategie, die sogenannte *Contrastive Examples* in $\mathcal{U}$ ausfindig macht. Contrastive Examples sind Datenpunkte, die zwar im Funktionsraum des Modells nahe beieinander liegen, sich jedoch stark in der vorhergesagten Wahrscheinlichkeitsverteilung unterscheiden. Margatina et al. [15] stellten die Hypothese auf, dass genau solche Contrastive Examples zu einem effizienteren Training des Machine Learning-Modells beitragen können.
Zunächst wird dafür das *Similarity Criterion* für die Datenpunkte x_i und x_j als

$$d\left(\Phi(x_i), \Phi(x_j)\right) < \epsilon$$

definiert [15]. Es sei $\Phi(\cdot) \in \mathbb{R}^{d'}$ ein Encoder, der x_i, x_j in einem gemeinsamen Funktionsraum darstellt, $d(\cdot)$ eine Abstandsfunktion und ϵ ein kleiner Wert für die Distanz. Nach diesem Kriterium ähneln sich zwei Datenpunkte x_i und x_j, wenn die Distanz zwischen diesen beiden Punkten kleiner als ϵ ist. Das zweite Kriterium ist als

$$KL\left(P(y|x_i)||P(y|x_j)\right) \rightarrow \infty$$

definiert [15]. Es seien $P(y|x_i)$ und $P(y|x_j)$ die Wahrscheinlichkeitsverteilungen der Datenpunkte x_i und x_j und KL die Kullback-Leibler Divergenz. Die Kullback-Leibler Divergenz gibt an, wie sehr sich zwei Wahrscheinlichkeitsverteilungen voneinander unterscheiden. Sie kann in diesem Kontext als Indikator für die Uncertainty des Modells betrachtet werden. Das zweite Kriterium besagt also, dass sich die Wahrscheinlichkeitsverteilungen der Punkte x_i und x_j maximal voneinander unterscheiden sollen. Die Kullback-Leiber Divergenz ist dabei als

$$KL\left(P||Q\right) = \sum_x P(x) \log(\tfrac{P(x)}{Q(x)})$$

definiert [15]. Dabei ist Q die Wahrscheinlichkeitsverteilung eines Datenpunktes aus dem *Unlabeled Pool* $\mathcal{U}$ und P die Wahrscheinlichkeitsverteilung eines Datenpunktes aus dem Trainingsdatensatz.
Margatina et al. beschreiben den Contrastive Active Learning-Loop anhand einer *Multi-Class Classification* mit C Klassen. Es soll dazu ein *Batch* Q aus b Datenpunkten aus dem Pool $\mathcal{U}$ extrahiert werden. Als Klassifikationsalgorithmus wurde das BERT Model $\mathcal{M}$ [6] verwendet. Das Classification Token von BERT diente dabei als Encoder $\Phi(\cdot)$ um die Datenpunkte der Pools $\mathcal{U}$ und $\mathcal{L}$ repräsentieren zu können.
Im ersten Schritt sollen die k nächsten Nachbarn mit einer *K-Nearest-Neighbor* Implementierung gefunden werden. Als Abstandsfunktion $d(\cdot)$ wurde die euklidische Distanz verwendet. Die Datenpunkte x lassen sich in $x_l \in \mathcal{L}$ und $x_p \in \mathcal{U}$ unterteilen. Für jeden Datenpunkt x_p wird dann eine Nachbarschaft $N_{x_p} = \{x_p, x_l^{(1)}..., x_l^{(k)}\}$ erstellt. Sie enthält den Punkt x_p und seine k nächsten Datenpunkte $x_l \in \mathcal{L}$.
Im nächsten Schritt wird ein sogenannter *Contrastive Score* s_{x_p} zwischen jedem Punkt x_p und seiner Nachbarschaft N_{x_p} berechnet. Dafür wird die Kullback-Leibler Divergenz zwischen x_p und all seinen Nachbarn $x_l \in N_{x_p}$ berechnet und anschließend der Durchschnitt der Divergenzen ermittelt. Ein hoher Contrastive Score s_{x_p} sagt also aus, dass der

Punkt x_p eine hohe Divergenz zu seinen Nachbarn aufweist. Anschließend werden alle Punkte $x_p \in \mathcal{U}$ in einem Ranking nach ihrem Contrastive Score s_{x_p} gelistet. Die obersten b Datenpunkte werden dann in das Batch Q übernommen und an das Orakel weitergegeben.

2.4 Abbruchkriterien

Abbruchkriterien können implementiert werden, um den Active Learning-Prozess unter bestimmten Bedingungen zu beenden. Das sogenannte *Kappa Average* [3] stoppt beispielsweise, wenn die Vorhersagen in Iteration t und $t + 1$ nur noch wenige Veränderungen vorweisen. Dafür wird eine Grenze für den *Agreement Score*, also die Übereinstimmung, übergeben. Das Kappa Average nutzt diese Grenze um zu entscheiden, ob die nächste Iteration des Active Learning-Verfahrens durchgeführt werden soll.
Als Abbruchkriterium wurde in dieser Arbeit eine feste Anzahl an Iterationen t gewählt, nach der das Active Learning-Verfahren abgebrochen werden soll. Damit sollte eine Basis zum Vergleich der verschiedenen Experimente nach einer bestimmten Anzahl an Iterationen geboten werden.

3 Klassifikationsalgorithmen

Die Wahl von geeigneten Algorithmen und Modellen zur Klassifikation von Texten hat einen großen Einfluss auf die Güte der Ergebnisse und sollte daher gut überlegt sein. In Active Learning-Szenarien kommt erschwerend hinzu, dass zum Training solcher Algorithmen nur ein kleiner Bruchteil der Daten zur Verfügung gestellt wird, da es eines der Ziele ist, Ressourcen wie Zeit und finanzielle Mittel einsparen zu können.
Mit der Veröffentlichung von auf Transformern [28] basierenden Modellen, wie zum Beispiel **BERT** [6] oder **GPT-2** [20], konnten qualitativ neue Maßstäbe für die Ausführung verschiedener NLP-Aufgaben gesetzt werden. Viele der traditionellen Klassifikationsalgortihmen basierten zuvor auf dem *TF-IDF (Term Frequency-Inverse Document Frequency) Maß*. Es setzt sich aus der *Term Frequency* und der *Inverse Document Frequency* zusammen und lässt sich als

$$tf\_idf(t, D) = tf(t, D) \cdot idf(t)$$

beschreiben. Dabei ist t der Term und D das zu untersuchende Dokument. Die Term Frequency $tf(t, D)$ gibt an, wie oft ein Term in einem bestimmten Dokument vorkommt. Für die Berechnung von $tf(t, D)$ gibt es verschiedene Ansätze. Einer davon ist das einfache Zählen des Terms t in D, also die Anzahl $\#(t,D)$. Häufig wird jedoch eine zusätzliche Normalisierung vorgenommen, die in der Berechnung die Länge eines Dokuments D mitberücksichtigt. Diese Term Frequency ist als

$$tf(t, D) = \frac{\#(t,D)}{max_{t' \in D} \#(t',D)}$$

definiert. Die zuvor beschriebene Normalisierung erfolgt hier anhand einer Division durch die Anzahl des am häufigsten vorkommenden Terms t' im selben Dokument D. Aus der normalisierten Berechnung ergeben sich Werte zwischen 0 und 1. Kommt ein Term gar nicht in einem Dokument vor, so erhält man eine 0 als Ergebnis. Ist der Term hingegen der am häufigsten vorkommende Term $max_{t' \in D}$, so erhält man eine 1 als Ergebnis der Berechnung.
Die Inverse Document Frequency $idf(t)$ gibt an, welche Relevanz ein bestimmter Term t im gesamten Textkorpus hat. Je seltener ein Term t insgesamt vorkommt, desto mehr Relevanz wird ihm zugeschrieben. Hierbei ist es wichtig noch einmal zu betonen, dass der Wert auf dem gesamten Textkorpus und nicht wie bei der Term Frequency nur auf einem Dokument berechnet wird. Die Inverse Document Frequency hingegen ist als

$$idf(t) = log(\frac{N}{\sum_{D:t \in D} 1})$$

definiert. Zunächst wird die Anzahl der Dokumente N durch die Anzahl der Dokumente, die den Term t beinhalten geteilt. Diese Berechnung ergibt mindestens 1, falls der Term in allen Dokumenten vorhanden ist und höchstens N, wenn der Term in nur einem Dokument D des gesamten Korpus auftritt. Anschließend wird eine Logarithmus-Funktion auf das Ergebnis angewandt, sodass die Inverse Document Frequency einer positiven Zahl, mindestens jedoch 0 entspricht. Je größer der Wert für die Inverse Document Frequency ist, desto spezifischer ist der Term. Das TF-IDF Maß ließe sich also beispielsweise

maximieren, indem ein Term t sehr häufig in nur einem einzigen Dokument D aus dem Textkorpus vorkommt.
Im Rahmen einer Klassifikationsaufgabe kann dieses Maß genutzt werden, um einen TF-IDF Vektor für jedes Dokument zu berechnen. Die Dimensionen des Vektors würden dann der Anzahl der verschiedenen Terme t im Korpus entsprechen. Es würde also für jedes Dokument das TF-IDF Maß für jeden im Textkorpus vorkommenden Term berechnet werden. Anschließend können diese Vektoren zusammen mit den annotierten Klassenlabels an den Klassifikationsalgortihmus übergeben werden. Als Beispiel kann hier eine *Support Vector Machine* genannt werden, welche diese Vektoren in einem Vektorraum darstellt. Unter Berücksichtigung der übergebenen Klassenlabels versucht die SVM dann eine möglichst geeignete Hyperebene zur Trennung der Klassen zu finden. Die gelernte Trennung der Datenpunkte mithilfe einer Hyperebene kann anschließend genutzt werden, um unbekannte Texte zu klassifizieren.
Diese Darstellung in Form eines TF-IDF Vektors hat jedoch den Nachteil, dass die kontextuellen Zusammenhänge der Textsequenzen nicht repräsentiert werden können. Dies ist einer der Gründe dafür, dass traditionelle Klassifikationsalgorithmen von Modellen wie BERT und GPT-2 übertroffen werden konnten. Hinzu kommt, dass moderne *language models* wie BERT von einem Pre-Training auf einem großen Datensatz profitieren, was ebenfalls einen markanten Einfluss auf das Sprachverständnis des trainierten Modells hat. Wie González-Carvajal et al. [9] zeigen konnten, übertrifft BERT die herkömmlichen Algorithmen wie Support Vector Machines oder Naive Bayes in Klassifikationsaufgaben auf dem englischsprachigen IMDB Datensatz [14]. Usherwood et al. [27] konnten ebenfalls bessere Ergebnisse mit BERT bei Klassifikationen auf verschiedenen Amazon und Twitter Datensätzen erzielen. Aufgrund der genannten Aspekte wird der Fokus dieser Arbeit daher auf die Klassifikation von Texten mithilfe von modernen Transformer-Modellen wie BERT und GPT-2 gelegt.

3.1 Das Transformer-Modell

Im Jahr 2017 präsentierten Vaswani et al. [28] die Architektur des sogenannten Transformer-Modells, die wie bereits erwähnt in vielen erfolgreichen NLP-Modellen Anwendung findet. Die Besonderheit an diesem Modell liegt dabei in der ausschließlichen Verwendung von *Attention* Mechanismen an Stelle von *Recurrent* oder *Convolutional Layern*. Dies führte zu neuen State-Of-The-Art Ergebnissen in elf verschiedenen NLP-Aufgaben, wie unter anderem der *WMT 2014 English-to-German Translation Task* und einer Verbesserung des *GLUE Score* [29].

3.1.1 Die Transformer-Architektur

Der Aufbau von Transformern basiert auf einer Encoder-Decoder Struktur. Der Encoder überführt den eingegebenen Text in eine numerische Repräsentation, die dazu dient mathematische Operationen zu ermöglichen. Im vorgestellten Modell besteht der Encoder aus $N = 6$ identischen Schichten (engl. Layer), die wiederum in je zwei Sublayer unterteilt werden. Dabei ist das erste Sublayer ein *Multi-Head Self-Attention* Mechanismus und das zweite Sublayer ein Feed-Forward Network (siehe linke Seite in Abbildung 4).

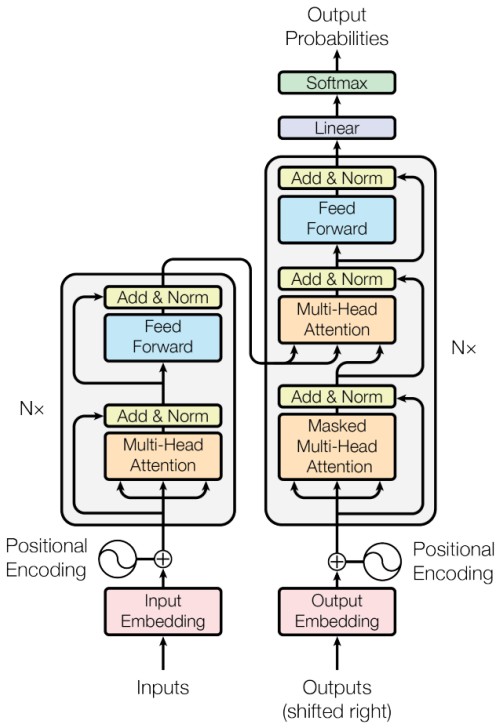

Abbildung 4: Architektur des Transformer-Modells [28]

Für jedes Sublayer wurde eine *Residual Connection* implementiert, die es ermöglicht bestimmte Schichten des neuronalen Netzes zu überspringen. Die Ausgabe jedes Sublayer ist daraufhin eine Normalisierung der vom Sublayer implementierten Funktion:

$$\text{LayerNorm}(x + \text{Sublayer}(x)).$$

Diese *Layer Normalization* [1] dient dazu, dem sogenannten *Covariate Shift* entgegenzuwirken. Der Covariate Shift kann als eine Verschiebung zwischen den Trainingsdaten und den tatsächlichen Daten beschrieben werden. Er ist also ein Indikator dafür, wie gut ein Machine Learning-Modell in der Lage ist, das gelernte Wissen zu generalisieren. Ba et al. beschreiben die Layer Normalization als eine Normalisierung über alle *Hidden Units* im selben Layer. Der Mittelwert μ^l und die Standardabweichung σ^l pro Layer l seien als

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

definiert [1]. Dabei ist H die Anzahl der Hidden Units im Layer und a_i^l die Vektorrepräsentation der aufsummierten Inputs der Neuronen im Layer l.
Der Decoder besteht ebenfalls aus $N = 6$ identischen Layern. Die Struktur des Decoders ähnelt stark dem Aufbau des zuvor beschriebenen Encoders, jedoch gibt es einen zusätzlichen Sublayer mit einem *Masked Multi-Head Attention* Mechanismus (Abbildung 4). Die Maskierung ist notwendig um zu gewährleisten, dass die Vorhersagen für Position i nur auf bereits bekanntem Output basieren. [2]

Der Attention Mechanismus
Der Attention Mechanismus wurde 2014 von Bahdanau et al. [2] vorgestellt. Zuvor verbreitete Modelle encodierten meist eine Inputsequenz in einen Vektor fester Länge, der wiederum vom Decoder entschlüsselt wurde. Dieser Problematik beugt der Attention Mechanismus vor, indem er nach relevanten Termen für die Vorhersage eines Wortes in der Inputsequenz sucht. Vaswani et al. beschreiben die Attention Funktion als Zuweisung einer *Query*, also einer Suchanfrage, und *Key-Value-Pairs*, also Schlüssel-Wert-Paaren, zu einem Output. Der Output entspricht dabei einer gewichteten Summe der Values [28]. Die Berechnung der Gewichtung für jeden Value erfolgt anhand einer Funktion, die die Query und den zugehörigen Key miteinbezieht. Diese Funktion wird als *Scaled Dot-Product Attention* bezeichnet, bei der die Queries und Keys die gleiche Dimension von d_k und Values eine Dimension von d_v haben. Die Attention ist mathematisch als

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

definiert. Die Berechnung mehrerer Queries erfolgt zeitgleich, da sie in einer Matrix Q zusammengeführt sind. Ebenso sind die Keys und Values jeweils in Form einer Matrix K und Matrix V dargestellt.

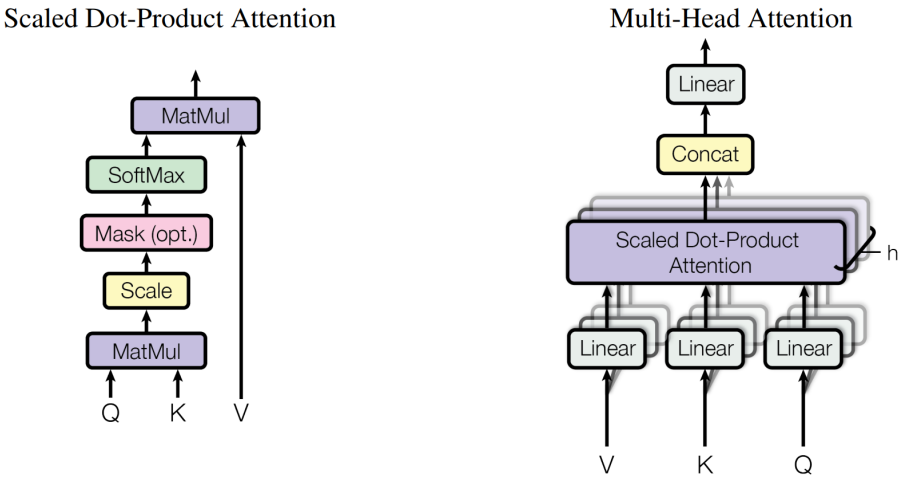

Abbildung 5: Scaled Dot-Product Attention (links), Multi-Head Attention (rechts) [28]

Zusätzlich sollte noch angemerkt werden, dass die Division durch $\sqrt{d_k}$ implementiert wurde, da sehr große Werte für d_k möglich sind. Bei der Division durch entsprechend große Werte für d_k würden der Softmax-Funktion nur sehr kleine Werte übergeben, was wiederum in einem sehr kleinen Gradienten resultieren würde.
Um diese Berechnung effizienter zu machen, wurde das Konzept der *Multi-Head Attention* eingeführt. Es profitiert von einer linearen Projektion der Queries, Keys und Values, die h-mal mit verschiedenen linearen Projektionen auf d_k, d_k und d_v Dimensionen durchgeführt wird. Dabei entspricht h der Anzahl der Attention Layer bzw. Heads. Anschließend wird auf jeder dieser Projektionen parallel die Attention-Funktion berechnet. Die

d_v-dimensionalen Output-Werte werden daraufhin konkateniert und wieder projiziert (siehe Abbildung 5). Das Konzept der *Multi-Head Self-Attention* wird mathematisch als

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^O$$
$$\text{mit}$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

beschrieben [28]. Die Projektionen sind hier die Parameter-Matrizen $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ und $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$. Vaswani et al. implementierten zur Umsetzung des Multi-Head Self-Attention Mechanismus $h = 8$ parallele Attention Heads und wandten sie auf die Vektoren mit den Dimensionen $d_k = d_v = d_{model} = 512$ an. So konnte die Dimension auf $d_k = d_v = d_{model}/h = 64$ pro attention head reduziert werden. [28]

Feed-Forward Networks
Wie in Abbildung 4 zu sehen ist, besitzt jedes Layer des Transformers ein *Feed-Forward Network*, welches separat auf jede Position angewandt wird. Dieses Network besteht aus zwei linearen Transformationen mit einer dazwischen liegenden *ReLU (Rectified Linear Unit) Activation Funktion*. Mathematisch kann das Feed-Forward Network als

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

beschrieben werden. Die ReLU Activation Funktion gibt dabei jeglichen positiven Input wieder als Output zurück. Sollte der Input jedoch negativ sein, so wird er zu einer 0 abgeändert. Die Matrizen W_1 und W_2 sind hier die *Weights* und b_1 und b_2 die entsprechenden *Biases*.

Die Softmax-Funktion
Eine *Softmax-Funktion* ermöglicht die Normalisierung eines Vektors z mit K reellen Zahlen in eine Wahrscheinlichkeitsverteilung mit K möglichen Ereignissen. Nach der Ausführung des Decoders werden die gelernten, linearen Transformationen mithilfe einer solchen Softmax-Funktion zu Wahrscheinlichkeiten für die Vorhersagen der Tokens umgewandelt. Die Softmax-Funktion ist als

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

definiert. Jeder der K Werte steht hier für eine der Klassen, deren Wahrscheinlichkeiten berechnet werden sollen. Die Werte des Vektors $z = (z_1, ..., z_K) \in \mathbb{R}^K$ durchlaufen dann jeweils eine Exponentialfunktion. Als Nächstes erfolgt eine Normalisierung, indem $exp(z_i)$ durch die Summe $\sum_{j=1}^{K} exp(z_j)$ geteilt wird. Nachdem die Softmax-Funktion angewandt wurde, liegen alle K Einträge (ein Eintrag für jede Klasse) zwischen 0 und 1 und summieren sich zu 1 auf. Sie können also als diskrete Wahrscheinlichkeitsverteilung interpretiert werden.

3.2 BERT

Eine der in dieser Arbeit verwendeten Methoden zur Textklassifikation basiert auf **BERT** [6], was für **B**idirectional **E**ncoder **R**epresentations from **T**ransformers steht. Der Aufbau von BERT kann in zwei wesentlichen Schritten beschrieben werden, dem *Pre-Training* und dem *Fine-Tuning*.

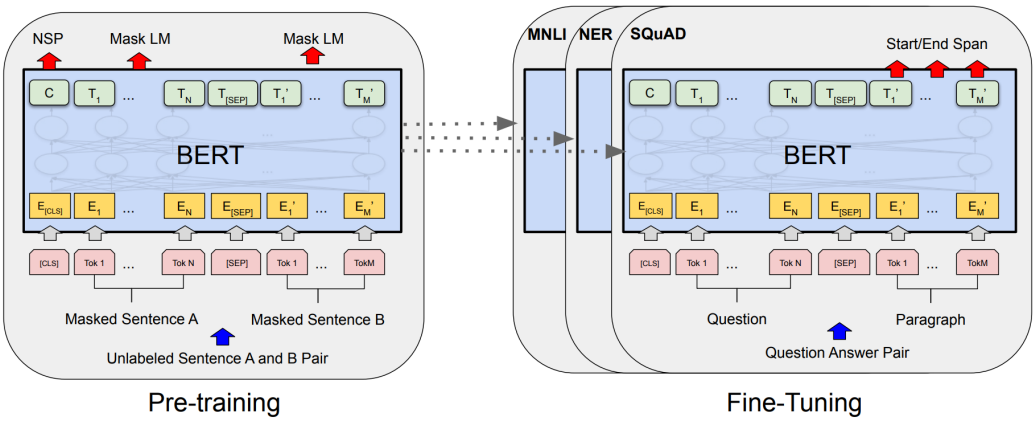

Abbildung 6: BERT Architektur [6]

Im ersten Schritt, dem sogenannten *Pre-Training*, findet ein *Unsupervised Learning*, also ein unüberwachtes Lernen, statt. Dabei werden verschiedene Aufgaben auf dem Datensatz ausgeführt, für die keine Labels oder Annotationen benötigt werden.
Im zweiten Schritt, dem *Fine-Tuning*, wird BERT ein Datensatz übergeben, der Labels enthält. Es findet also ein *Supervised Learning*, also überwachtes Lernen, statt. Dies ermöglicht es, BERT auf eine spezifische Aufgabe wie zum Beispiel Textklassifikationen oder das Beantworten von gestellten Fragen zu trainieren. Die Aufteilung in diese zwei Schritte hat den Vorteil, dass BERT sowohl die Eigenschaften und kontextuellen Strukturen der zum Pre-Training verabreichten Sprache lernt, als auch flexibel im Fine-Tuning auf verschiedene NLP-Aufgaben angepasst werden kann.

3.2.1 Die BERT-Architektur

BERT ist ein Multi-Layer Bidirectional Transformer Encoder, der auf der Transformer Implementierung von Vaswani et al. [28] basiert. Es gibt eine Unterteilung in die Modelle BERT_{BASE} und BERT_{LARGE}. Die beiden Modelle unterscheiden sich hauptsächlich in ihrer Größe. BERT_{LARGE} ist das Größere der beiden Modelle und erreicht in vielen NLP-Aufgaben bessere Ergebnisse als BERT_{BASE}. BERT_{BASE} erfordert hingegen weniger Rechenleistung, was es sehr praktikabel macht. Aufgrund der besseren Effizienz wurde in dieser Arbeit ausschließlich das BERT_{BASE} Model verwendet. Es besteht aus $L = 12$ Layern, einer Hidden Size $h = 768$, $A = 12$ Attention Heads und hat insgesamt $110M$ Parameter. Die Größe von BERT_{BASE} wurde dabei bewusst so gewählt, dass ein Vergleich zum GPT Transformer Model gegeben ist, welches jedoch im Gegensatz zu BERT nicht bidirektional ist.

3.2.2 BERT's Input Repräsentation

Für die interne Repräsentation nutzt BERT verschiedene Arten von *Embeddings*. Diese lassen sich in *WordPiece Embeddings* [31], *Segment Embeddings* und *Position Embeddings* unterteilen. Zusätzlich wird die Textsequenz intern um mehrere Tokens erweitert.

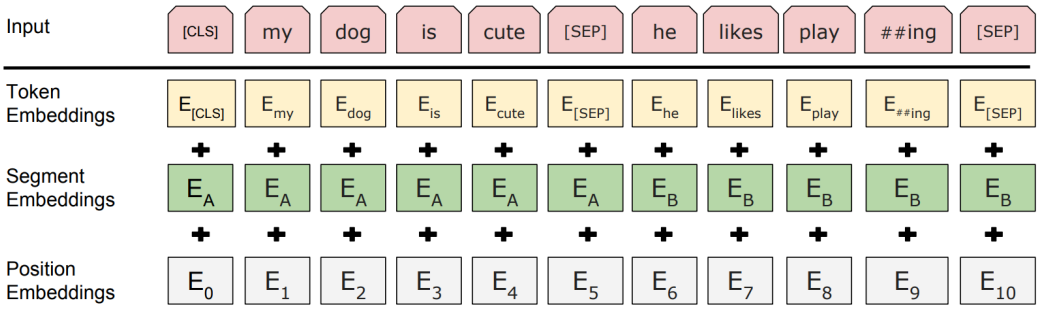

Abbildung 7: BERT's Input Repräsentation [6]

Der Beginn einer Sequenz wird mit einem *Classification Token* [CLS] vor der Sequenz gekennzeichnet (siehe Abbildung 7). Dabei können einzelne Sätze oder Satzpaare als eine Sequenz betrachtet werden. Der *Final Hidden State* des Classification Tokens entspricht der zusammengesetzten Repräsentation der gesamten Sequenz, weshalb er für Klassifikationsaufgaben genutzt werden kann. Es sollte zusätzlich angemerkt werden, dass Devlin et al. zusammenhängende Teile des Textes als Satz bezeichnen. Dies ist in einigen Fällen, jedoch nicht immer mit einem linguistischen Satz gleichsetzbar. Das *Separation Token* [SEP] wird jeweils am Ende von Satz A und Satz B eingefügt. Zusätzlich wird ein gelerntes *Segmentation Embedding* für jedes Token hinzugefügt. Es gibt für jedes Token darüber Auskunft, ob ein Token zu Satz A oder Satz B zugehörig ist. Gehört ein Token zu Satz A, so wird es beispielsweise mit dem Segmentation Embedding E_A dargestellt. Das Position Embedding enthält wichtige Informationen über die Position eines Tokens in der Sequenz. So hat jedes Token an Position i ein zugehöriges Embedding E_i. Die *Input Embeddings* ergeben sich anschließend aus der Summe der WordPiece Embeddings, Segment Embeddings und Position Embeddings.

3.2.3 WordPiece Embeddings

Um einen Text in eine interne Repräsentation zu übertragen, macht BERT vom Konzept der *Word Embeddings* Gebrauch [31]. Word Embeddings sind eine *Feature Vector Representation* eines Wortes und werden anhand eines Kontexts berechnet. Die Grundannahme ist, dass Wörter, die in einer semantisch engen Relation zueinander stehen auch in einem ähnlichen Kontext gemeinsam auftreten. Als Kontext eines Wortes wird dabei der Bereich um das Wort herum bezeichnet. Es wird also jeweils pro Wort eine Reichweite an Wörtern vor und nach diesem Wort betrachtet.

Die Darstellung im Vektorraum ermöglicht es, Ähnlichkeiten von Worten anhand von Distanzen zwischen den Datenpunkten auszudrücken. In Abbildung 8 sieht man, dass

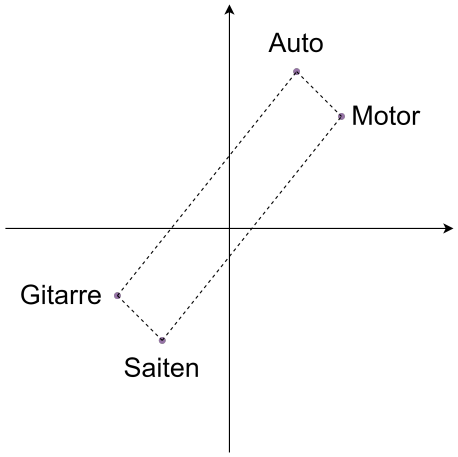

Abbildung 8: Word Embeddings Beispiel

die Worte *Auto* und *Motor* sehr nahe beieinander liegen. Eine ähnlich kurze Distanz haben auch die Worte *Gitarre* und *Saiten*. Allerdings besteht eine größere Distanz zwischen den Worten *Auto* und *Gitarre* sowie zwischen den Worten *Motor* und *Saiten*. Dem liegt zugrunde, dass die Wortpaare (*Auto*, *Motor*) und (*Gitarre*, *Saiten*) häufiger in einem gemeinsamen Kontext auftreten, als die Wortpaare (*Gitarre*, *Auto*) und (*Saiten*, *Motor*).
Wu et al. präsentierten im Jahr 2016 eine besondere Form von Word Embeddings, die sogenannten *WordPiece Embeddings* [31]. Wie der Name bereits vermuten lässt, werden die Worte dabei in *Sub-word Units* zerlegt. Diese können als Bestandteile von Worten in Form von Buchstaben, Silben oder auch als gesamtes Wort auftreten. BERT nutzt diese WordPiece Embeddings mit einem Vokabular von 30 000 Tokens. Der Vorteil ist hierbei insbesondere eine Verbesserung des Verständnisses für Worte, die nicht im Pre-Training enthalten waren. Denn unbekannte Worte bestehen häufig aus einer Kombination von bekannten Sub-word Units.

Jet makers feud over seat width with big orders at stake

_J et _makers _fe ud _over _seat _width _with _big _orders _at _stake [31]

Anhand des Beispielsatzes lässt sich erkennen, wie Sub-word Pieces aufgebaut sind. Der Beginn eines Wortes wird immer mit dem Token „_“ dargestellt. Desweiteren wurden einige Worte wie zum Beispiel „Jet“ in mehrere Sub-Word Pieces „_J“ und „et“ aufgeteilt. Andere Worte wie „makers“ und „seat“ wurden hingegen nicht in mehrere Sub-Word Pieces unterteilt, sondern bekamen nur den Anfangs-Token „_“ als Präfix.

3.2.4 Pre-Training

Das Pre-Training von BERT basiert auf zwei bidirektionalen, unüberwachten Lernaufgaben (siehe Abbildung 6). Die erste Aufgabe ist das *Masked Language Model (MLM)*, gefolgt von der *Next Sentence Prediction (NSP)*. Sowohl das MLM als auch die NSP wurden auf

einer Kombination aus dem *BooksCorpus Datensatz* [32] mit 800 Mio. Worten und einem Wikipedia Datensatz mit 2500 Mio. Worten durchgeführt. Der Wikipedia Datensatz wurde auf die reinen Textdaten reduziert. Um dies zu erreichen, wurden Tabellen, Listen, Bilder und ähnliche Inhalte daraus entfernt. Beide Datensätze sind englischsprachig und monolingual. Es wurde kein *Shuffling* (Vermischung) der beiden Datensätze betrieben, um möglichst lange, semantisch zusammenhängende Textsequenzen für das Training beizubehalten.
In dieser Arbeit wurde zur Verbesserung des deutschsprachigen Sprachverständnisses das *German BERT Model* (kurz GBERT) [5] verwendet. Es wurde zu in der Pre-Training Phase mit einem 12GB großen, deutschen Datensatz trainiert. Dieser setzt sich aus Wikipedia Daten (6 GB), *Open Legal Data* (2,4 GB) [16] und Zeitungstexten (3,6 GB) zusammen.

Masked Language Model (MLM)

Das Ziel beim MLM ist es, Vorhersagen für Tokens anhand des Kontexts zu treffen. Da BERT jedoch ein bidirektionales Modell ist, sind die Tokens links und rechts von der aktuellen Position bereits bekannt. Dies ist problematisch, da die Vorhersage dadurch beeinflusst werden würde, dass BERT das korrekte Ergebnis bereits kennt. Die Lösung für dieses Problem ist eine Maskierung der Textsequenz. Devlin et al. maskieren daher 15% aller WordPiece Tokens in jeder Sequenz. Diese werden zufällig ausgewählt und sollen anschließend von BERT vorhergesagt werden.
Für die Maskierung wird ein *Mask Token* ([MASK]) verwendet, um das ursprüngliche Token zu ersetzen. Im Hinblick auf die Tatsache, dass beim Fine-Tuning keine Mask Token vorhanden sind, werden nicht die gesamten 15% aller WordPiece Tokens durch dieses Token ersetzt. Stattdessen wird in 80% der Maskierungen das tatsächliche Mask Token eingesetzt, in 10% ein zufälliges anderes Token und in den restlichen 10% bleibt das Token an Position i unverändert. Die *Final Hidden Vectors* $T_i \in \mathbb{R}^H$ der entsprechenden Mask Tokens durchlaufen dann eine Softmax-Funktion über das gesamte WordPiece Vokabular. Anschließend wird T_i benutzt, um mithilfe einer *Cross Entropy Loss* Funktion das ursprüngliche Token vorherzusagen. Der Cross Entropy Loss ist eine sogenannte *Loss Function*. Solche Funktionen werden im Machine Learning genutzt, um die Fehlerrate bei Modellen zu bemessen. Dabei liegen die Werte des Cross Entropy Loss zwischen 0 und 1. Je kleiner der Wert ist, desto besser ist das Modell. Dementsprechend ist es das Ziel des Modells, diesen Wert zu minimieren damit möglichst akkurate Vorhersagen getroffen werden können.

Next Sentence Prediction (NSP)

Zusätzlich zum MLM wird eine Next Sentence Prediction durchgeführt. Bei der NSP werden jeweils zwei Sätze A und B untersucht. Die Aufgabe ist es herauszufinden, ob Satz B der Nachfolger von Satz A ist. Dazu wird eine Textabfolge erstellt, bei der Satz B in 50% der Fälle der wirkliche Nachfolger von Satz A ist. Bei den restlichen 50% wurde ein zufälliger Satz aus dem Korpus hinter A positioniert, der nicht der Nachfolger von Satz A ist. Als Nächstes wird ein Vektor C für die Vorhersage des nächsten Satzes verwendet (siehe Abbildung 6). Die Ausführung von NSP im Pre-Training hat den Hintergrund, dass BERT nicht nur Zusammenhänge zwischen den einzelnen Tokens, sondern auch mehr über die Relationen zwischen den Sätzen lernt. Dies ist ein wichtiger

Bestandteil des Sprachverständnisses und führt insbesondere bei NLP-Aufgaben, die auf Satzebene agieren (z.B. Question Answering), zu einer Verbesserung der Ergebnisse.

3.2.5 Fine-Tuning

Das Fine-Tuning von BERT ist sehr flexibel und lässt sich auf mehrere NLP-Aufgaben anwenden. Dafür werden je nach Aufgabenstellung die entsprechenden Inputs und Outputs an BERT übergeben.

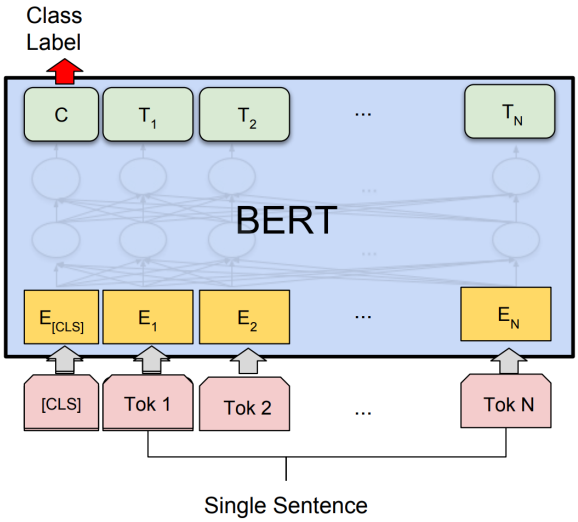

Abbildung 9: Fine-Tuning von BERT für eine Klassifikationsaufgabe [6]

Bei einer Textklassifikation werden beispielsweise die zu klassifizierenden Texte und ihre zugehörigen Klassenlabels übergeben. Auf Grundlage dieser Daten findet dann ein überwachtes Lernen statt. Für die Klassifikation wird dann die Repräsentation des Classification Tokens [CLS] an einen Output Layer übergeben (siehe Abbildung 9).

3.3 DistilBERT

Die stetig wachsende Größe von *Pre-Trained Language Models* der letzten Jahre (siehe Abbildung 10) hat zu neuen State-Of-The-Art Ergebnissen im Bereich des Natural Language Processing geführt. Jedoch stellt sich zunehmend die Frage, wie man solche Modelle effizienter gestalten kann, sodass sie schneller in der Anwendung sind und weniger Rechenleistung erfordern. Einen Ansatz dafür liefert *DistilBERT* [22]. DistilBERT ist eine Abwandlung von BERT, die mithilfe von Kompressionstechniken die Größe des Modells reduziert und dabei versucht möglichst vergleichbare Ergebnisse zu erzeugen.
Sanh et al. konnten mit DistilBERT die Größe des BERT Modells um 40% reduzieren, während immer noch 97% des Sprachverständnisses (bemessen am GLUE Score [29]) erhalten werden konnten. Zusätzlich ist DistilBERT 60% schneller als BERT, was es in der Praxis sehr effizient macht.

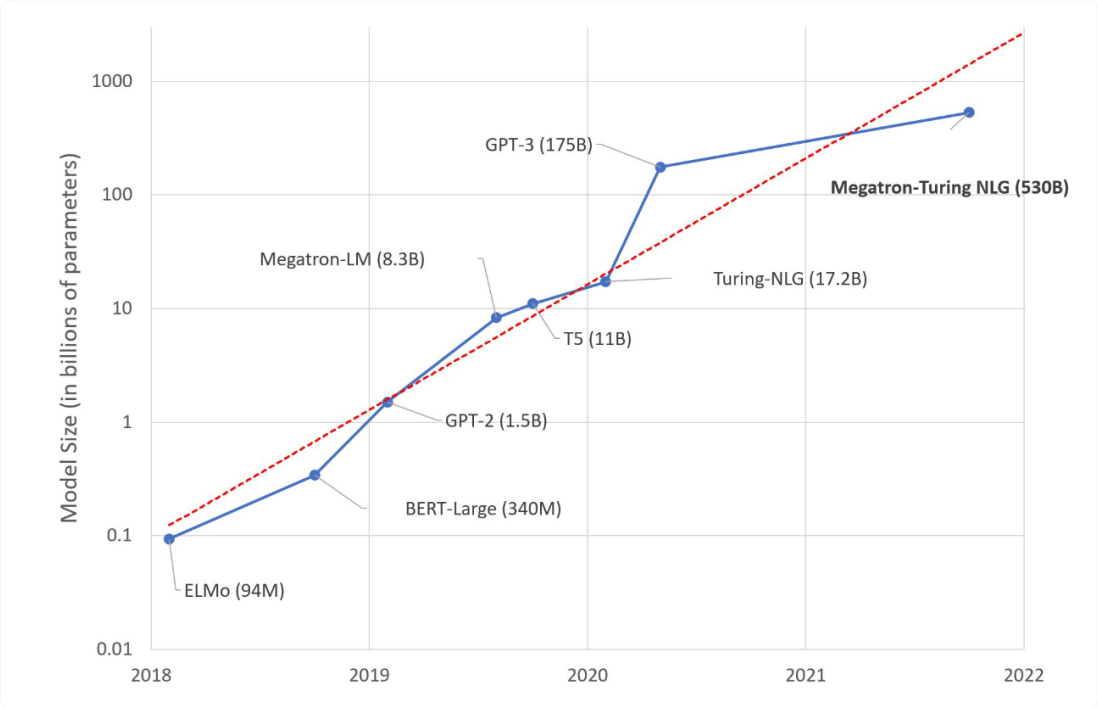

Abbildung 10: Größe verschiedener Pre-Trained Language Modelle der letzten Jahre [26]

3.3.1 Knowledge Distillation

Um diese Ergebnisse erreichen zu können, wurde eine Technik namens *Knowledge Distillation* [4] angewandt. Das Grundprinzip besteht darin, ein kleineres Modell (*Student*) von einem größeren Modell (*Teacher*) lernen zu lassen, um dessen Verhalten reproduzieren zu können. Dafür werden drei Loss Funktionen definiert, die anschließend für die Optimierung des Trainings verwendet werden.
Der Student wird zunächst mit einem *Distillation Loss* über die Wahrscheinlichkeiten des Teachers trainiert.

$$L_{ce} = \sum_i t_i \cdot log(s_i)$$

Hier sind t_i die Wahrscheinlichkeit des Teachers und s_i die Wahrscheinlichkeit des Students für Klasse i. Der Wert s_i wird für jede Klasse i logarithmiert, mit t_i multipliziert und anschließend wird eine Summe aus den Ergebnissen gebildet.
Angenommen t_i hat Klasse i eine recht hohe Wahrscheinlichkeit zugewiesen und s_i weist derselben Klasse nur einen sehr geringen Wert zu. Da s_i logarithmiert wird, entsteht bei der Berechnung des Beispiels ein sehr hoher, negativer Zahlenwert. Durch die multiplikative Verknüpfung von t_i und $log(s_i)$ und das Aufsummieren dieser Werte für jede Klasse trägt sich dieser negative Wert in der Berechnung fort. Bei einer großen Diskrepanz zwischen s_i und t_i entsteht also ein recht hoher, negativer Zahlenwert. Daraufhin wird eine Abwandlung der Softmax-Funktion implementiert, die *Softmax-Temperature*. Sie ist als

$$p_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}$$

definiert [10]. In der Berechnung ist z_i der *Model Score* für Klasse i und mit dem Hyperparameter T lässt sich die *Smoothness*, also die Glättung, der Funktion beeinflussen. Das Ergebnis p_i ist die Wahrscheinlichkeit für die zugehörige Klasse i. Bei der ursprünglichen Softmax-Funktion ist $T = 1$. DistilBERT nutzt einen Wert $T > 1$ für das Smoothing der Wahrscheinlichkeitsverteilungen der Klassen. Während des Trainings wird derselbe Wert für T im Teacher und Student Model angewandt. Bei der Inferenz, also dem produktiven Einsatz des Transformer Models, wird $T = 1$ gesetzt, was wieder der ursprünglichen Softmax-Funktion entspricht.
Das Modell wird dann mithilfe einer Linearkombination aus dem *Distillation Loss* L_{ce} und dem *Masked Language Model Loss* L_{mlm} [6] trainiert. Zusätzlich wurde noch ein *Cosine Embedding Loss* L_{cos} hinzugefügt, der ein Indikator dafür ist, wie gut die Richtungen der Hidden State Vectors von Teacher und Student miteinander übereinstimmen.

3.3.2 Die Architektur von DistilBERT

Die Architektur von DistilBERT ähnelt sehr stark der Architektur von BERT, jedoch wurden einige Elemente entfernt. So sind die *Token-Type Embeddings* und das *Pooling Layer* nicht in DistilBERT vorhanden. Zusätzlich wurde die Anzahl der Layer bei DistilBERT halbiert. Aufgrund der halbierten Layer-Anzahl wurde zur Initialisierung des Student Modells immer eines von zwei Layern aus dem Teacher Model übernommen. Desweiteren wurde die Next Sentence Prediction Task aus dem Pre-Training Schritt entfernt und DistilBERT auf dem selben Datensatz wie BERT trainiert.

3.4 GPT-2

Im Jahr 2019 veröffentlichten Radford et al. die zweite Generation des ***G**enerative **P**re-**T**rained Transformer* Modells [20]. GPT-2 hat insgesamt $1{,}5$ Mrd. Parameter während BERT_{LARGE} vergleichsweise nur 340 Mio. Parameter hat (siehe Abbildung 10). Radford et al. testeten GPT-2 in einem *Zero-Shot Setting* und erreichten neue State-Of-The-Art Ergebnisse bei sieben von acht *Language Modeling Datasets*. Als Zero-Shot Setting wird dabei ein Testverfahren bezeichnet, bei dem der *Learner* Datenpunkte Klassen zuordnen muss, denen er nicht zuvor im Training begegnet ist. Genau wie BERT basiert GPT-2 auf dem von Vaswani et al. vorgestellten Transformer Model [28]. Jedoch nutzt BERT die Encoder Blöcke des Transformer Models, während GPT-2 die Decoder Blöcke verwendet. Auch bei GPT-2 gibt es eine Aufteilung in eine Pre-Training und eine Fine-Tuning Phase.

3.4.1 Die Architektur von GPT-2

Die Architektur von GPT-2 kann als Modifikation des GPT Modells betrachtet werden. Da sich die Architekturen stark ähneln, ist es sinnvoll zunächst die Architektur von GPT zu beschreiben. Die GPT Architektur basiert zum Großteil auf dem Transformer Modell.

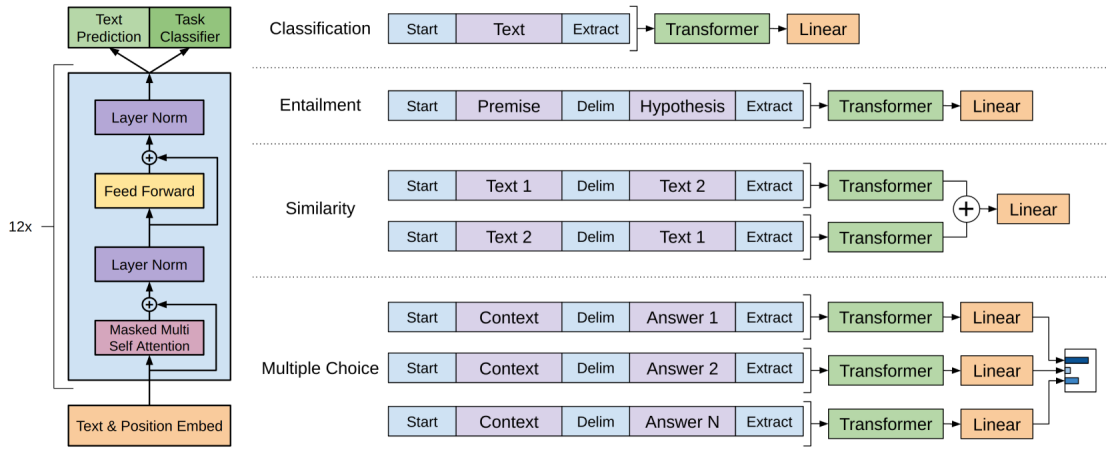

Abbildung 11: Transformer Architektur in GPT (links), Input Transformations beim Fine-Tuning (rechts) [19]

GPT wurde auf einem *12-Layer-Decoder-Only Transformer* mit zwölf Masked Self-Attention Heads pro Layer trainiert. Die Feed-Forward Networks haben 3072-dimensionale *Inner States*. Es wurde ein *Adam Optimizer* mit einer maximalen *Learning Rate* von $2{,}5e-4$ implementiert. Der Adam Optimizer dient dazu, optimale Werte für die *Weights* zu finden, um den Loss zu minimieren. Nach den Masked Multi Self-Attention Layern und den Feed-Forward Networks folgt jeweils eine Layer Normalization (siehe Abbildung 11). Das Modell wurde für 100 Epochen auf 64 zufällig gewählten, kontinuierlichen Sequenzen mit einer Größe von je 512 Tokens trainiert. Desweiteren werden sogenannte *Bytepair Encodings* eingesetzt, die GPT helfen sollen mit *Out Of Vocabulary* Tokens umzugehen. Es wurde eine *L2 Regularization* verwendet, um einem möglichen *Over-Fitting* entgegenzuwirken. Als Over-Fitting wird eine Überanpassung des Modells auf dem Trainingsdatensatz bezeichnet. L2-Regularization verschafft bei diesem Problem Abhilfe, indem es die Koeffizienten quadriert und als *Penality* zur Loss Function hinzufügt. Dies hat den Effekt, dass Koeffizienten das Ergebnis der Loss Function stärker beeinflussen je größer sie sind. GPT nutzt eine *Gaussian Error Linear Unit* (GELU) als Activation Function. Diese ähnelt stark der von BERT genutzten RELU, jedoch sind kleine negative Werte möglich (siehe Abbildung 12). [19]
Bei GPT-2 wurde die Layer Normalization zum Input jedes Sub-Blocks verschoben und eine zusätzliche Layer Normalization wurde nach dem letzten Self Attention Block eingefügt. Die Größe der kontinuierlichen Sequenzen für das Training wurde von 512 auf 1024 Tokens angehoben und die Batchsize auf 512 erhöht. Außerdem wurde das Vokabular des Bytepair Encoding von $40{,}000$ auf $50{,}257$ Merges erhöht.

3.4.2 Das Bytepair Encoding

Das *Bytepair Encoding* (BPE) wurde bereits 1994 von Gage et al. vorgestellt [8] und war ursprünglich als Kompressionsalgorithmus konzipiert. Die Grundidee dieser

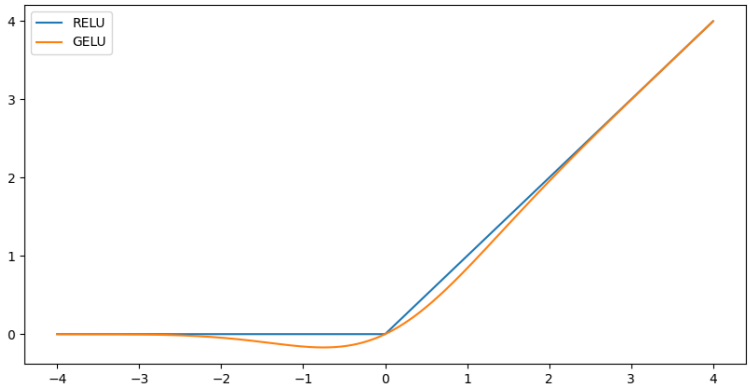

Abbildung 12: GELU und RELU im Vergleich (x-Achse: Input, y-Achse: Output)

Kompression ist eine Reduktion eines Datensatzes auf Basis von frequent auftretenden Sequenzen. Dafür wurden die am häufigsten vorkommenden Bytepairs iterativ durch ein einzelnes Byte ersetzt, das nicht im Datensatz vorkam. Dieses Konzept kann auch im Rahmen einer Wortsegmentierung verwendet werden, indem statt Bytes *Characters* (Buchstaben) verwendet werden.
Sennrich et al. wandten dieses Prinzip im Jahr 2015 im Kontext eines *Neural Machine Translation* Modells an [24]. Das Ziel war es, die Qualität von Übersetzungen zwischen verschiedenen Sprachen mithilfe von Machine Learning-Modellen zu verbessern. Eines der Hauptprobleme stellten *Out Of Vocabulary Words* dar. Dies sind Wörter, die dem Modell nicht bekannt sind, da sie nicht während des Trainings aufgetreten sind. Ähnlich wie bei BERT ist der Lösungsansatz für dieses Problem eine Aufteilung des Inputs in Sub-Word Units. Betrachtet man beispielsweise das Wort *Finanzmarktstabilisierung*, so wird schnell deutlich warum eine Repräsentation in Form von Sub-Word Units von Vorteil sein kann. Denn nimmt man das Wort als Ganzes, so tritt es vermutlich sehr selten auf und ist mit hoher Wahrscheinlichkeit nicht im Trainingsdatensatz enthalten. Teilt man es jedoch auf in *Finanz* | *markt* | *stabilisierung*, so erhält man Sub-Word Units, die häufiger auftreten. Um eine geeignete Aufteilung der Wörter in Sub-Word Units zu generieren, nutzten Sennrich et al. Bytepair Encoding. Zunächst wird dafür ein Vokabular auf Character-Level erstellt und an jedes Wort ein spezielles „· “ Token konkateniert, das das Ende eines Wortes symbolisiert. Dieses End-Token „· “ dient dazu, das Originalwort anschließend wiederherstellen zu können.

Da die Textsequenz nun als eine Abfolge von Charactern betrachtet wird, beginnt der Algorithmus in Abbildung 13 mit dem Zählen der aufeinanderfolgenden Character-Paare. Dies ist in der Funktion „get_stats “ beispielhaft aufgeführt. Die korrekte Bezeichnung für eine aufeinanderfolgende Character-Sequenz lautet *Character* n*-Gram*. Es werden zunächst *Character Uni-Grams* betrachtet, also Character n-Grams mit $n = 1$. Auf dem Character Uni-Gram Paar mit der größten Anzahl an Vorkommnissen findet dann ein *Merge* statt.

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

Abbildung 13: Bytepair Encoding Beispielcode in Python [24]

Bei einem Merge werden die beiden Character Uni-Grams dann zusammengeführt und daraufhin als ein Symbol betrachtet. Dieser Vorgang wird mehrfach wiederholt, woraufhin ein größeres Vokabular entsteht. In späteren Iterationen können auch Character n-Grams mit $n \geq 1$ auftreten und in einem Merge Schritt zusammengeführt werden. Die Größe dieses Vokabulars ist die Summe aus dem ursprünglichen Vokabular und der Anzahl der durchgeführten Merges. Im Beispielcode ist das End-Token als <\w> dargestellt. In Abbildung 13 wird beispielhaft der Algorithmus in Anwendung auf die Worte { low, lower, newest, widest } demonstriert. Die zuvor gelernten Bytepair Encodings entstammen aus dem folgenden Vokabular { low, lowest, newer, wider }.

r ·	→	r·
l o	→	lo
lo w	→	low
e r·	→	er·

Abbildung 14: Bytepair Encoding Beispielcode [24]

Die beiden Vokabulare sehen zunächst sehr ähnlich aus, allerdings war beispielsweise das Wort *lower* während der Trainingsphase nicht vorhanden. Mithilfe der zuvor gelern-

ten BPE kann das Wort *lower* jedoch aus den Sub-Word Units „low “ und „er· “ zusammengesetzt werden (siehe Abbildung 14). [24]

3.4.3 Pre-Training

Für das Pre-Training von GPT-2 wurde ursprünglich der englischsprachige BooksCorpus Datensatz [32] verwendet. Er besteht aus mehr als 7000 unveröffentlichten Büchern verschiedener Genres wie zum Beispiel Krimi, Horror oder Fantasy. Dieser Datensatz wurde für das Training gewählt, weil Bücher in der Regel lange, kontinuierliche Textabschnitte beinhalten. So kann das Transformer Modell mithilfe des Attention Mechanismus Abhängigkeiten auf großer Reichweite lernen. Der BooksCorpus Datensatz [32] kam auch beim Pre-Training von BERT in Kombination mit anderen Datensätzen zum Einsatz.

3.4.4 Fine-Tuning

Das Fine-Tuning von GPT wurde mit den gleichen Hyperparametern wie im Pre-Training durchgeführt. Der *Classifier* hatte dabei eine *Dropout Rate* von $0{,}1$ und es wurden eine *Learning Rate* von $6{,}25e-5$ und eine *Batchsize* von 32 vewendet. Da das Modell sich im Fine-Tuning schnell anpasst, reichte eine Anzahl von 3 Epochen bereits aus. [19]
Wie bereits erwähnt, wurde GPT-2 primär auf Zero-Shot Tasks angewandt, sodass kein Fine-Tuning vorgenommen wurde [20].

4 Der Raddialog-Datensatz

Der in dieser Arbeit verwendete Datensatz stammt aus einem Online-Partizipationsverfahren der Städte Bonn, Moers und dem Kölner Stadtteil Ehrenfeld. Im hier beschriebenen Partizipationsverfahren, dem sogenannten *Raddialog*, wurde den Bürgern eine Online-Plattform zur Verfügung gestellt, auf der sie sich kritisch in Bezug auf den Radverkehr in den jeweiligen Städten äußern konnten. Dieses Internetportal war im September und Oktober des Jahres 2017 zugänglich und ermöglichte das Erstellen von Textbeiträgen, die anschließend zu einem Datensatz aggregiert wurden.
Der daraus resultierende Datensatz beinhaltet insgesamt 3139 Beiträge, davon sind 2314 aus Bonn, 459 aus Moers und 366 aus Ehrenfeld.

4.1 Verteilung der Primärkategorien mit Einzelzuweisungen

Diese Beiträge, bestehend aus einem Titel und einem Textkommentar in deutscher Sprache, wurden in die folgenden acht Kategorien unterteilt: *Radverkehrsführung*, *Radwegqualität*, *Hindernisse*, *Ampeln*, *Beschilderung*, *Fahrradparken*, *Beleuchtung* und *Sonstiges*. Die Nutzer der Online-Plattform mussten beim Erstellen eines Beitrages eine dieser acht sogenannten *Primärkategorien* auswählen. Der folgende Beitrag stammt aus dem Bonner Raddialog und wurde der Primärkategorie *Fahrradparken* zugeordnet:

Stellplätze

Hier wie fast in der gesamten Innenstadt, vor allem auch Sternstraße, fehlen vernünftige Abstellmöglichkeiten für Räder. Das sollten höhere Ständer sein, an die man Räder anlehnen und an denen man sie auch vernünftig anketten kann.

Im Durchschnitt bestand ein Beitrag aus 4,07 Sätzen und 66,83 Worten. Um eine bessere Datenqualität gewährleisten zu können, haben zunächst die Moderatoren des Internetportals eine Prüfung der Zuweisungen und bei Bedarf Korrekturen der Zuweisungen vorgenommen. Im Anschluss hat ein Analyst die kategorischen Einordnungen der Beiträge geprüft und falls notwendig weitere Anpassungen vorgenommen. Ursprünglich wurde jedem Beitrag genau eine Kategorie zugewiesen, jedoch kam es in einigen Fällen vor, dass ein Beitrag mehreren Kategorien zuweisbar war. Aufgrund dessen beinhaltet der Datensatz sowohl eine Annotation, die jedem Beitrag exakt eine Primärkategorie zuweist, als auch eine Annotation, die mehrere Primärkategorien pro Beitrag ermöglicht. Tabelle 1 zeigt die Verteilung der Primärkategorien mit Einzelzuweisung des gesamten Datensatzes auf. Diese Verteilung kann wiederum für die Städte Bonn, Moers und Ehrenfeld aufgeteilt werden. Zunächst soll der aggregierte Datensatz aus der Spalte *Gesamt* betrachtet werden. Es wird ersichtlich, dass die Kategorie *Radverkehrsführung* mit 1437 Beiträgen am stärksten vertreten ist und mit 45,78% fast die Hälfte des gesamten Datensatzes ausmacht. Die am zweitstärksten vertretene Klasse *Radwegqualität* hat bereits nur noch 618 Repräsentanten und macht mit 19,69% knapp ein Fünftel des Datensatzes aus.

	Gesamt		Bonn		Moers		Ehrenfeld	
	Anzahl	Anteil in %	Anzahl	Anteil in %	Anzahl	Anteil in %	Anzahl	Anteil in %
Radverkehrsführung	1437	45,78	1020	44,08	222	48,37	195	53,28
Radwegqualität	618	19,69	449	19,40	111	24,18	58	15,85
Hindernisse	385	12,27	319	13,79	31	6,75	35	9,56
Ampeln	259	8,25	178	7,69	47	10,24	34	9,29
Beschilderung	185	5,89	150	6,48	19	4,14	16	4,37
Fahrradparken	139	4,43	108	4,67	9	1,96	22	6,01
Sonstiges	68	2,17	53	2,29	10	2,18	5	1,37
Beleuchtung	48	1,53	37	1,60	10	2,18	1	0,27
Summe	3139	100	2314	100	459	100	366	100

Tabelle 1: Verteilungen der Primärkategorien mit Einzelzuweisung pro Stadt und Gesamt

Dies ist im Gesamtvergleich immer noch ein relativ hoher Wert, jedoch nimmt die Anzahl im Vergleich zur am stärksten vertretenen Klasse bereits markant ab. So macht die am drittstärksten vertretene Klasse *Hindernisse* mit $12{,}27\%$ lediglich knapp ein Achtel des Datensatzes aus. Am schwächsten vertreten ist die Klasse *Beleuchtung* mit nur 48 Repräsentanten, was $1{,}53\%$ entspricht.
Als Nächstes erfolgt die Gegenüberstellung der Datensätze aus den Städten Bonn, Moers und Köln-Ehrenfeld. Bei Betrachtung von Tabelle 1 fällt direkt auf, dass die Verteilungen der jeweiligen Städte sehr homogen sind. In Bonn, Moers und Ehrenfeld steht die *Radverkehrsführung* an erster Stelle und macht immer mindestens $44{,}08\%$ des Datensatzes aus. Grundsätzlich ähneln sich die kategorischen Verteilungen der Städte sehr stark und es gibt diesbezüglich nur vereinzelte Ausnahmen. Beispielsweise gab es in Moers mehr Beiträge zur Kategorie *Ampeln* als zur Kategorie *Hindernisse*, wohingegen es in Bonn und Ehrenfeld umgekehrt war.

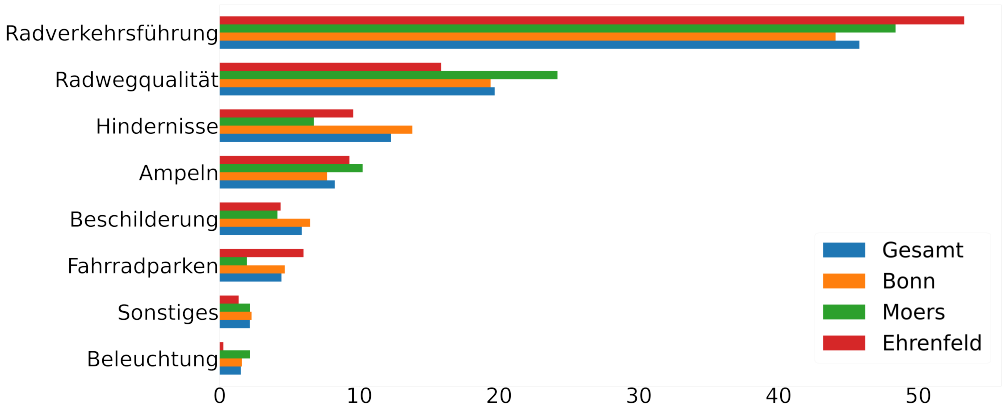

Abbildung 15: Balkendiagramm: Verteilung der Primärkategorien (x-Achse: Anteil der Beiträge in Prozent, y-Achse: Name der Primärkategorie)

Zusammenfassend gilt es festzustellen, dass die Verteilungen der acht Primärkategorien stark ungleichmäßg sind, was anhand von Abbildung 15 nochmal verdeutlicht wird. Vergleicht man die Kategorien paarweise, so fällt auf, dass auch die Kategorien im mittleren Vorkommensbereich Differenzen in ihrer Repräsentationsstärke aufweisen.

4.2 Verteilung der Primärkategorien mit Mehrfachzuweisungen

Wie bereits erwähnt gab es eine weitere Annotation, bei der mehrfache Zuweisungen der Primärklassen pro Beitrag zulässig waren. Die zugehörige Verteilung der Primärklassen bei dieser Annotation wird in Tabelle 2 dargestellt.

	Gesamt		Bonn		Moers		Ehrenfeld	
	Anzahl	Anteil in %	Anzahl	Anteil in %	Anzahl	Anteil in %	Anzahl	Anteil in %
Radverkehrsführung	1490	42,52	1057	41,26	229	44,99	204	47,11
Radwegqualität	708	20,21	519	20,26	118	23,18	71	16,40
Hindernisse	442	12,61	364	14,21	33	6,48	45	10,39
Ampeln	287	8,19	197	7,69	51	10,02	39	9,01
Beschilderung	229	6,54	182	7,10	27	5,30	20	4,62
Fahrradparken	147	4,20	112	4,37	9	1,77	26	6,00
Sonstiges	137	3,91	84	3,28	27	5,30	26	6,00
Beleuchtung	64	1,83	47	1,83	15	2,95	2	0,46
Summe	3504	100	2562	100	509	100	433	100

Tabelle 2: Verteilungen der Primärkategorien pro Stadt und Gesamt

Insgesamt wurden den 3139 Beiträgen mit 3504 Labels zugewiesen. Auch hier fällt auf, dass die Klassenverteilung sehr stark imbalanciert ist. Dabei ist die *Radverkehrsführung* mit $42{,}52\%$ weiterhin die am stärksten repräsentierte Klasse. Sie hat bei den einzelnen Städten jeweils einen Anteil von mindestens 40%. Die Primärkategorie *Beleuchtung* hingegen ist am schwächsten repräsentiert und macht in jedem der Datensätze nur weniger als 2% aus. In Ehrenfeld wurde diese Klasse sogar nur zweimal annotiert und machte damit nur $0{,}46\%$ des Ehrenfelder Datensatzes aus.

4.3 Verteilung der Sekundärkategorien

Da innerhalb der acht Primärkategorien eine große inhaltliche Diversität vorhanden war, wurden sie zusätzlich in 30 *Sekundärkategorien* unterteilt, bei denen ebenfalls Mehrfachzuweisungen zulässig waren.

Primärkategorie	Sekundärkategorie
Radverkehrsführung	Vorschlag für neuen Radweg Unklare Verkehrsführung für Radfahrende Sichere Straßenquerung fehlt Mangelnde Sichtbeziehungen Fahrradstraße einrichten Regelwidriges Verhalten Einbahnstraße für Radverkehr öffnen Radweg beidseitig befahren Radwegbenutzungspflicht überprüfen Auffahrt auf Radweg nur mit Umweg möglich Geschwindigkeitsbegrenzung
Radwegqualität	Zu geringe Breite Unebenheit durch Brüche oder Risse Übergänge mit zu großen Höhenunterschieden Wiederholt Schmutz oder Wasser auf Radweg
Ampeln	Ampelschaltung ungünstig Ampel(-ergänzung) vorschlagen Ampel entfernen
Hindernisse	Radweg permanent zugeparkt Behinderung durch feste Gegenstände Radweg häufig blockiert
Sonstiges	Nicht ortsgebundene Vorschläge Sonstige Hinweise Mängelmeldung
Beschilderung	Fahrbahnmarkierung Radweg fehlt oder schlecht s. Radwegweisung fehlt oder schlecht sichtbar
Beleuchtung	Beleuchtung fehlt Falsche Beleuchtung
Fahrradparken	Keine oder zu wenig Abstellmöglichkeiten Ungeeignete Abstellanlagen

Tabelle 3: Zuweisungen von Primärkategorien zu Sekundärkategorien

Wie in Tabelle 3 dargestellt, werden jeder Primärkategorie zwischen zwei und elf Sekundärkategorien zugewiesen, welche wieder auf die acht ursprünglichen Primärklassen zurückführbar sind. Die am stärksten vertretene Primärklasse *Radverkehrsführung* besitzt auch die größte inhaltliche Diversität und konnte in insgesamt elf Sekundärkategorien wie zum Beispiel *Vorschlag für einen neuen Radweg* und *Mangelnde Sichtbeziehungen* aufgeteilt werden (siehe Tabelle 3). Die am zweitstärksten vertretene Primärklasse *Radwegqualität* ist bereits nur noch in vier Sekundärklassen eingeteilt worden, während die restlichen Primärklassen jeweils zwei bis drei Unterkategorien aufwiesen.

	Gesamt	B.	M.	E.
Vorschlag für neuen Radweg	601	394	120	87
Unebenheit Brüche oder Risse	329	222	75	32
Zu geringe Breite	302	237	36	29
Radweg permanent zugeparkt	257	206	14	37
Unklare Verkehrsführung für Radfahrende	255	207	37	11
Sichere Straßenquerung fehlt	207	140	28	39
Ampelschaltung ungünstig	184	122	46	16
Fahrbahnmarkierung Radweg fehlt oder schlecht s.	147	112	15	20
Behinderung durch feste Gegenstände	131	108	15	8
Keine oder zu wenig Abstellmöglichkeiten	130	95	9	26
Mangelnde Sichtbeziehungen	107	82	16	9
Nicht ortsgebundene Vorschläge	104	59	22	23
Fahrradstrasse einrichten	93	74	4	15
Ampel(-ergänzung) vorschlagen	92	68	6	18
Regelwidriges Verhalten	87	74	9	4
Radwegweisung fehlt oder schlecht sichtbar	84	71	13	0
Einbahnstraße für Radverkehr öffnen	74	37	5	32
Übergänge mit zu großen Höhenunterschieden	72	49	13	10
Radweg häufig blockiert	67	62	5	0
Wiederholt Schmutz oder Wasser auf Radweg	64	48	11	5
Radweg beidseitig befahren	60	48	12	0
Beleuchtung fehlt	56	42	12	2
Radwegebenutzungspflicht überprüfen	45	31	7	7
Geschwindigkeitsbegrenzung	38	23	0	15
Sonstige Hinweise	32	23	5	4
Auffahrt auf Radweg nur mit Umweg möglich	24	24	0	0
Ungeeignete Abstellanlagen	19	19	0	0
Mängelmeldung	15	11	4	0
Ampel entfernen	13	8	0	5
Falsche Beleuchtung	8	5	3	0
Summe	3697	2701	542	454

Tabelle 4: Anzahl der Repräsentanten pro Sekundärkategorie und Stadt (B. = Bonn, M. = Moers, E. = Ehrenfeld)

In Tabelle 4 ist die Verteilung der Sekundärkategorien dargestellt. Aufgrund der möglichen Mehrfachzuweisungen gibt es insgesamt 3697 Zuweisungen für die Sekundärkategorien. Die Kategorie *Vorschlag für neuen Radweg* macht mit einer Anzahl von 601 Annotationen fast ein Sechstel des gesamten Datensatzes aus. Wie man in Tabelle 3 erkennen kann, ist die Kategorie *Vorschlag für neuen Radweg* der Primärkategorie *Radverkehrsführung* zuzuordnen, welche wiederum den Großteil der Primärkategorien ausmachte. Auch bei den Sekundärkategorien ist demnach bermerkbar, dass einige Klassen sehr

stark und andere sehr schwach vertreten sind. Ein Indikator dafür ist die große Differenz zwischen der Kategorie *Vorschlag für neuen Radweg* mit 601 Beiträgen und der am zweitstärksten vertretenen Kategorie *Unebenheit Brüche oder Risse* mit 329 Beiträgen. Die ungleiche Verteilung wird besonders deutlich, wenn man sich vor Augen führt, dass die Kategorie *Vorschlag für neuen Radweg* mit 601 Repräsentanten sogar stärker vertreten ist als die 14 am schwächsten vertretenen Kategorien in Summe mit 587 Repräsentanten. Die am schwächsten repräsentierte Klasse *Falsche Beleuchtung* wurde insgesamt nur in acht Fällen annotiert, was gerade einmal 0,22% des gesamten Datensatzes entspricht.
Desweiteren gab es insgesamt 495 Beiträge mit Mehrfachzuweisungen auf der Ebene der Sekundärkategorien. Insgesamt enthielt der Datensatz 196 verschiedene Kombinationen von Klassen. Davon bestanden 144 aus zwei Kategorien, 44 aus drei Kategorien und acht aus mehr als drei Kategorien. Aufgrund des stark imbalancierten Datensatzes, der großen Anzahl an Klassen und der teilweise sehr geringen Repräsentationsstärke ist die thematische Klassifikation der hier beschriebenen Sekundärkategorien als besonders schwierige Aufgabe einzustufen.

5 Evaluation und Vergleich

5.1 Das Setting der Experimente

Die im Rahmen dieser Arbeit vorgenommenen Experimente wurden mithilfe der Python Bibliothek *Small-Text* durchgeführt [23]. Diese enthält mehrere hilfreiche Funktionen für die Implementierung eines Active Learning-Verfahrens wie zum Beispiel Initialisierungsstrategien, Query Strategien oder Abbruchkriterien. Zusätzlich ist eine Integration der von *Huggingface* [30] zur Verfügung gestellten Transformer Modelle wie BERT und GPT-2 möglich. Wie bereits in Kapitel 4 angemerkt wurde, existieren verschiedene Annotationen für den Raddialog-Datensatz. Es gab eine Unterscheidung zwischen den Primär- und Sekundärklassen. Die Annotation der Primärklassen kann weitergehend in Einzel- und Mehrfachzuweisungen eingeteilt werden, während bei den Sekundärklassen immer Mehrfachzuweisungen zulässig waren. Aufgrund der hohen Anzahl an Sekundärklassen wurde die Trainingsdatenmenge für dieses Setting gegenüber den Primärklassen verdoppelt.
In beiden Settings kam zunächst eine *Stratified Random Initialization* [7] zum Einsatz. Für die Klassifikation mit acht Klassen wurden initial 150 Samples gezogen, bei der Klassifikation mit 30 Klassen waren es hingegen 300 Samples. In jedem Experiment wurde entweder BERT, DistilBERT oder GPT-2 verwendet. Es folgte die Auswahl einer Query Strategie. Dabei wurde entweder Random Sampling, Breaking Ties, Prediction Entropy oder Contrastive Active Learning verwendet. Jede der Query Strategien hat dabei jeweils 10 Queries ausgeführt. Im einfacheren Setup mit acht Primärklassen wurden pro Iteration 30 Samples gezogen, im komplexeren Fall mit 30 Klassen wurden je Iteration 60 Samples gezogen. In den folgenden Experimenten soll jede Kombinationsmöglichkeit der in dieser Arbeit vorgestellten Klassifikationsalgorithmen und Query Strategien untersucht werden.

Es wurde für jedes Experiment eine sogenannte *K-Fold Cross Validation* durchgeführt.

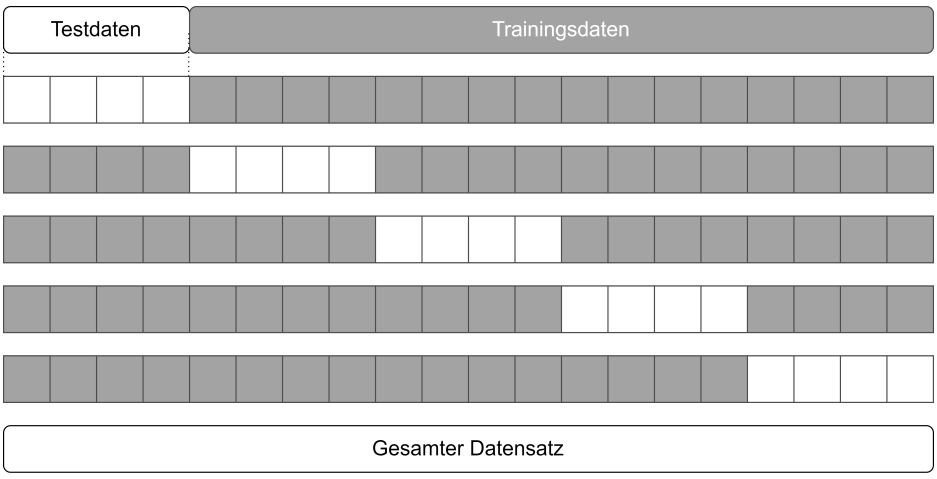

Abbildung 16: Beispiel einer 5-Fold Cross Validation

Bei der K-Fold Cross Validation erfolgt eine Aufteilung des Gesamtdatensatzes in k gleich große Test- und Trainingsdatensätze. In Abbildung 16 ist ein Beispiel mit $k = 5$ Splits dargestellt. Bei einer 5-Fold Cross Validation wird jeweils auf 80% der Daten trainiert und auf 20% der Daten getestet. Führt man diesen Prozess fünfmal mit jeweils disjunkten Testdatensätzen durch, so hat man nach k Ausführungen jeden Datenpunkt des Gesamtdatensatzes einmal getestet. Die Ergebnisse werden anschließend gesammelt und können dann ausgewertet werden.

Die K-Fold Cross Validation ist insbesondere bei dem hier vorliegenden Datensatz mit nur 3139 Textbeiträgen von großer Relevanz. Denn bei einem einfachen 80%-20%-Split ohne K-Fold Cross Validation würden nur knapp 628 Datenpunkte zum Testen verwendet werden. Dies ist hier besonders problematisch, da der stark imbalancierte Datensatz zur Folge haben könnte, dass einige der Klassen nicht in der Testmenge vorkommen würden. Um eine zuverlässigere Evaluation gewährleisten zu können, wurde daher eine 5-Fold Cross Validation für jedes Experiment durchgeführt.
Zusätzlich wurde stichprobenartig eine Zeitmessung für die Dauer jedes Experimentes durchgeführt, um zu evaluieren wie praxisfähig die jeweiligen Verfahren sind. Diese Zeitmessungen können variieren und sind vom verwendeten System abhängig. Alle Experimente wurden auf demselben Endgerät mit einer *NVIDIA Geforce GTX 1070* Grafikkarte durchgeführt. Sie hat eine Standard-Speicherkonfiguration von *8GB GDDR5* und besitzt 1920 *NVIDIA CUDA Cores*.

5.2 Gütemaße der Evaluation

5.2.1 F_1-Score

Der F1-Score ist das harmonische Mittel von *Precision* p und *Recall* r und soll hier als primärer Indikator für die Güte des Modells verwendet werden.

$$F_1 = \frac{2}{\frac{1}{p}+\frac{1}{r}}$$

Die Definition von Precision und Recall lässt sich am einfachsten anhand einer binären Klassifikation mit einer positiven und einer negativen Klasse erklären. Dabei werden die Datenpunkte der positiven Klasse als *Positives* und die der negativen Klasse als *Negatives* bezeichnet. Klassifiziert das Modell einen Datenpunkt korrekterweise als Positive, so wird dies *True Positive* t_p genannt. Wird ein Datenpunkt als Positive klassifiziert, obwohl es eigentlich ein Negative ist, so bezeichnet man dies als *False Positive* f_p. Die Definition von *True Negatives* t_n und *False Negatives* f_n ist analog dazu.

5.2.2 Precision

Die Precision p setzt die True Positives t_p mit einer Summe der True Positives und False Positives $t_p + f_p$ in ein Verhältnis.

$$p = \frac{t_p}{t_p+f_p}$$

Das Ergebnis ist dabei ein Wert zwischen 0 und 1. Angenommen alle als Positive klassifizierten Datenpunkte sind korrekt, also True Positives, so sind die False Positives $f_p = 0$ und es ergibt sich eine Precision von $p = 1$.

5.2.3 Recall

Der Recall r hingegen setzt die True Positives t_p mit einer Summe aus True Positives und False Negatives $t_p + f_n$ in ein Verhältnis.

$$r = \frac{t_p}{t_p+f_n}$$

Somit gibt der Recall an, wie vollständig die Daten klassifiziert wurden. Werden also alle Elemente einer Klasse bei der Klassifikation erkannt, so hat diese Klasse einen Recall von $r = 1$.

Der F_1-Score als harmonisches Mittel dieser beiden Werte ist aufgrund der Wechselwirkung dieser beiden Maße äußerst sinnvoll. Angenommen es gibt 1000 Positives und das Machine Learning-Modell klassifiziert nur eines dieser Positives als solches. Es wird kein anderes Element als Positive klassiziert, also ist $f_p = 0$ und es ergibt sich für diese Klasse eine Precision von $p = 1$. Obwohl nur ein Element der Klasse korrekt klassifiziert wurde, ist die Precision der Positives maximal, da es keine False Positives gibt. Betrachtet man jedoch den Recall, so ergibt sich nur ein Wert von $r = \frac{1}{1000}$, da nur ein Element der Klasse korrekt klassifiziert wurde.
Ein weiteres Beispiel wäre ein Modell, das jeden Datenpunkt als Positive klassifiziert. Für die Positives würde sich der Recall $r = 1$ ergeben, da jedes Element der Klasse auch als solches klassifiziert wurde. Die Precision wäre jedoch deutlich geringer, da alle Negatives des Datensatzes als False Positives klassifiziert werden würden. Aufgrund dieser Wechselwirkung zwischen Precision und Recall stellt der F_1-Score ein robustes und aussagekräftiges Maß für die Güte des Modells dar.
Als Durchschnittswert soll hier der *Micro Average F_1-Score* dienen. Er kann als globaler F_1-Score betrachtet werden, da beim Micro Average F_1-Score die Summen der True Positives, False Positives, True Negatives und False Negatives aller Klassen gebildet werden. Diese Summen werden dann für die anschließende Berechnung von Precision, Recall und den resultierenden Micro Average F_1-Score verwendet.

5.2.4 Accuracy

In der Active Learning-Literatur ist es gängig die *Accuracy* für jede Iteration des Verfahrens in Form eines Graphen anzugeben. Sie wird in der Regel auf dem Testdatensatz berechnet und kann somit als Test Accuracy bezeichnet werden. Die Accuracy a ist als

$$a = \frac{t_p+t_n}{t_p+t_n+f_p+f_n}$$

definiert. Dabei werden die korrekt klassifizierten True Positives t_p und True Negatives t_n zunächst aufsummiert. Daraufhin erfolgt eine Division der Ergebnisse durch die gesamte

Anzahl der getesteten Datenpunkte, also $t_p + t_n + f_p + f_n$. Angenommen es werden 10 Datenpunkte getestet und es werden $t_p + t_n = 6$ Datenpunkte korrekt klassifiziert. Die restlichen 4 Datenpunkte jedoch werden nicht korrekt klassifiziert, also gilt $f_p + f_n = 4$. Setzt man dies nun in die Accuracy-Formel ein, so ergibt sich die Accuracy $a = \frac{6}{6+4} = \frac{6}{10} = 0{,}6$.

5.3 Primärklassen mit Einzelzuweisungen

5.3.1 BERT - Primärklassen mit Einzelzuweisung

Anhand des in Abbildung 17 dargestellten Graphen lässt sich erkennen, wie schnell die verschiedenen Query Strategien dem BERT Modell halfen sich anzupassen.

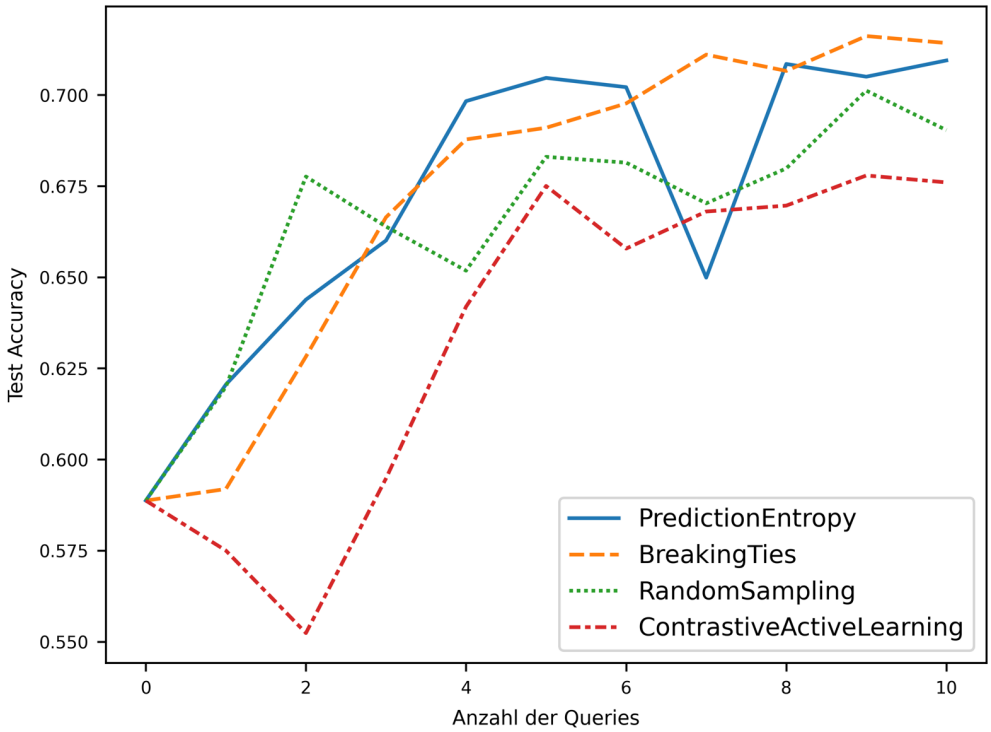

Abbildung 17: Übersicht: Accuracy aller Query Strategien pro Query (BERT)

Es ist erkennbar, dass das Random Sampling in den Iterationen 1 und 2 einen starken Anstieg der Accuracy hatte. Jedoch wurde es bereits in Iteration 3 von der Prediction Entropy und Breaking Ties übertroffen. Contrastive Active Learning erzielte fast durchgehend eine niedrigere Accuracy als die anderen Query Strategien. Mithilfe der Prediction Entropy konnte bereits nach vier Queries eine Accuracy von über $0{,}70$ erreicht werden, jedoch gab es in Iteration 7 einen Einbruch der Accuracy. Ab Query 8 stieg die Accuracy bei der Prediction Entropy jedoch wieder steil an. Am Ende hatte Breaking Ties die höchste Accuracy, dicht gefolgt von der Prediction Entropy. Es ist auffällig, dass fast alle Lernkurven zwischendurch gewisse Einbrüche in der Accuracy hatten. Bei der Breaking Ties Strategie waren diese Einbrüche jedoch wesentlich geringer als bei den anderen Query Strategien.

Wie man anhand von Tabelle 5 erkennen kann, erreichte *Breaking Ties* den höchsten F_1-Score von $0{,}70$.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,00	**0,76**	0,75	0,08	48
Sonstiges	0,00	0,00	**0,18**	0,00	68
Fahrradparken	0,86	**0,88**	0,82	0,27	139
Beschilderung	0,15	0,42	**0,46**	0,36	185
Ampeln	0,75	0,75	**0,76**	0,74	259
Hindernisse	0,58	**0,62**	0,59	0,55	385
Radwegqualität	0,66	**0,69**	0,64	0,68	618
Radverkehrsführung	**0,77**	**0,77**	**0,77**	**0,77**	1437
micro avg	0,66	**0,70**	0,69	0,65	3139

Tabelle 5: Vergleich der F_1-Scores aller Query Strategien (BERT)

Als Referenzwert kann hier das Training von BERT auf dem gesamten Trainingsdatensatz betrachtet werden. Dabei erreichte BERT einen F_1-Score von $0{,}73$. Durch den Einsatz von Active Learning konnte hier also über 95% der Güte des Modells bei gleichzeitiger Reduktion des annotierten Trainingsdatensatzes erhalten werden. Breaking Ties konnte bei vier der acht Primärklassen den höchsten F_1-Score erzielen. Alle Query Strategien erzielten bei der *Radverkehrsführung* einen F_1-Score von $0{,}77$, also gab es für diese Klasse keinen klaren Favoriten. Die Prediction Entropy erreichte mit einem Wert von $0{,}69$ bei der *Radwegqualität* den zweithöchsten F_1-Score. Prediction Entropy erzielte insgesamt bei drei der acht Primärklassen die besten F_1-Scores. Das Contrastive Active Learning war hier mit einem F_1-Score von $0{,}65$ weniger erfolgreich als das Random Sampling mit $0{,}66$.

Die Zeitmessungen haben ergeben, dass BERT in Kombination mit Random Sampling mit einer Zeit von 06:57 Minuten am schnellsten durchgeführt werden konnte (siehe Tabelle 6).

Query Strategie	Zeitmessung
Random Sampling	06:57
Breaking Ties	09:11
Prediction Entropy	08:58
Contrastive Active Learning	09:42

Tabelle 6: Vergleich: Zeitmessung nach Query Strategien (BERT)

Das Klassifikation mit Contrastive Active Learning hat mit einer Zeit von 09:42 Minuten am längsten gedauert. Die beiden Query Strategien mit den höchsten F_1-Scores, also Breaking Ties und Prediction Entropy, haben 09:11 Minuten und 08:58 Minuten benötigt.

5.3.2 DistilBERT - Primärklassen mit Einzelzuweisung

Abbildung 18 zeigt, dass die Prediction Entropy in Kombination mit DistilBERT fast durchgehend die höchste Accuracy aufwies.

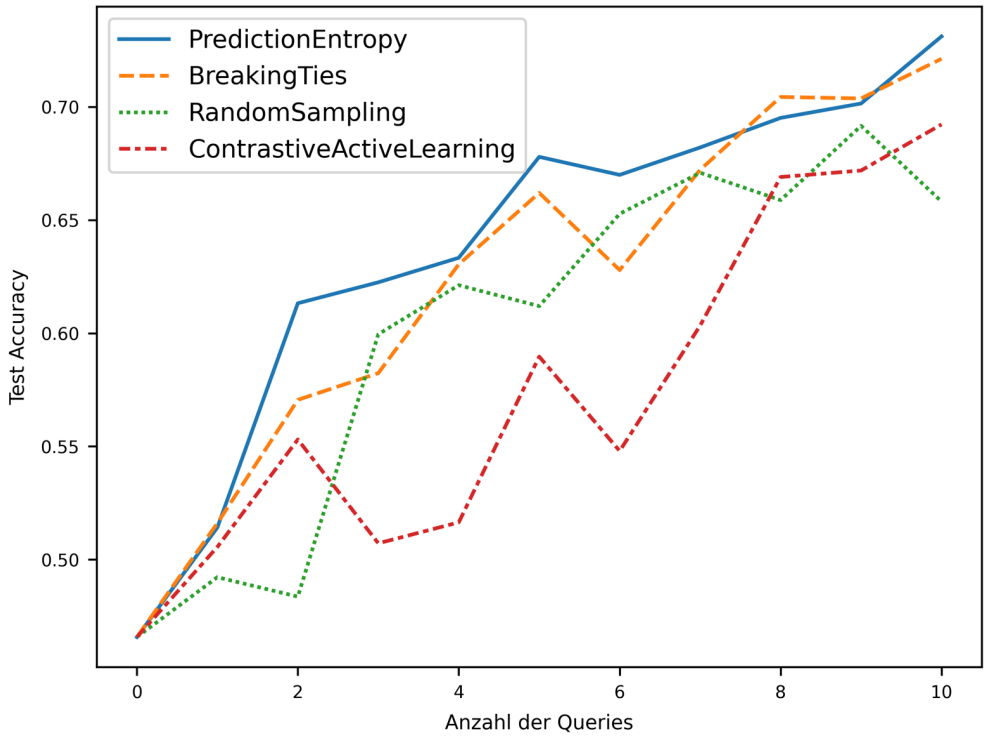

Abbildung 18: Übersicht: Accuracy aller Query Strategien pro Query (DistilBERT)

Die Prediction Entropy wurde nur zwischenzeitlich (nach Query 8 und 9) knapp von der Breaking Ties Strategie übertroffen. Sowohl die Prediction Entropy als auch Breaking Ties waren hauptsächlich die zwei führenden Query Strategien in Bezug auf die Accuracy. Das Contrastive Active Learning hatte fast durchgehend die geringste Accuracy und lag die meiste Zeit sogar unterhalb der Kurve des Random Sampling. Beim Contrastive Active Learning fällt insbesondere auf, dass es zwischen den Queries 6 bis 8 größere Sprünge nach oben gemacht hat. Nach der letzten Iteration übertraf das Contrastive Active Learning knapp das zufallsbasierte Random Sampling.

Aus Tabelle 7 geht hervor, dass die Prediction Entropy in Kombination mit DistilBERT einen F_1-Score von $0{,}72$ und damit die besten Ergebnisse vorwies. Der Referenzwert bei einem Training mit DistilBERT auf dem gesamten Trainingsdatensatz lag bei $0{,}73$ für den F_1-Score. Daher konnte in diesem Setting sogar über 98% der Güte des Modells im Active Learning-Setting erhalten werden.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,00	0,69	**0,70**	0,42	48
Sonstiges	0,00	0,00	0,00	0,00	68
Fahrradparken	0,34	0,90	**0,91**	0,67	139
Beschilderung	0,06	0,40	**0,48**	0,39	185
Ampeln	0,72	0,74	**0,78**	0,72	259
Hindernisse	0,43	**0,65**	**0,65**	0,52	385
Radwegqualität	0,66	**0,68**	**0,68**	**0,68**	618
Radverkehrsführung	0,77	0,78	**0,79**	0,77	1437
micro avg	0,61	0,71	**0,72**	0,67	3139

Tabelle 7: Vergleich der F_1-Scores aller Query Strategien (DistilBERT)

Die Prediction Entropy erreichte bei allen acht Primärklassen den höchsten F_1-Score oder einen Gleichstand mit den anderen Query Strategien. Sie ist jedoch dicht gefolgt von der Breaking Ties Strategie mit einem F_1-Score von $0{,}71$. Bei der Klasse *Hindernisse* gab es einen Gleichstand mit einem F_1-Score von $0{,}65$ für die Prediction Entropy und Breaking Ties. Für die Klasse *Radwegqualität* erzielten Breaking Ties, Prediction Entropy und Contrastive Active Learning einen F_1-Score von $0{,}68$. Der F_1-Score der Klasse *Sonstiges* ist hier bei allen Query Strategien 0 gewesen.

Aus Tabelle 8 geht hervor, dass alle Active Learning-Verfahren mit DistilBERT jeweils in weniger als 7 Minuten durchgeführt werden konnten und somit sehr schnell waren.

Query Strategie	Zeitmessung
Random Sampling	04:41
Breaking Ties	05:44
Prediction Entropy	05:38
Contrastive Active Learning	06:43

Tabelle 8: Vergleich: Zeitmessung nach Query Strategien (DistilBERT)

Das Random Sampling war hier mit 04:41 Minuten am schnellsten, lieferte jedoch bemessen am F_1-Score die schlechtesten Ergebnisse. Das Contrastive Active Learning hat mit 06:43 am längsten gebraucht und lieferte lediglich mittelmäßige Ergebnisse. Die Query Strategien Breaking Ties und Prediction Entropy waren beide in knapp unter 6 Minuten fertig und lieferten hier die besten F_1-Scores.

5.3.3 GPT-2 - Primärklassen mit Einzelzuweisung

Für GPT-2 ergaben sich bei den Lernkurven der verschiedenen Query Strategien teilweise sehr starke Schwankungen (siehe Abbildung 19).

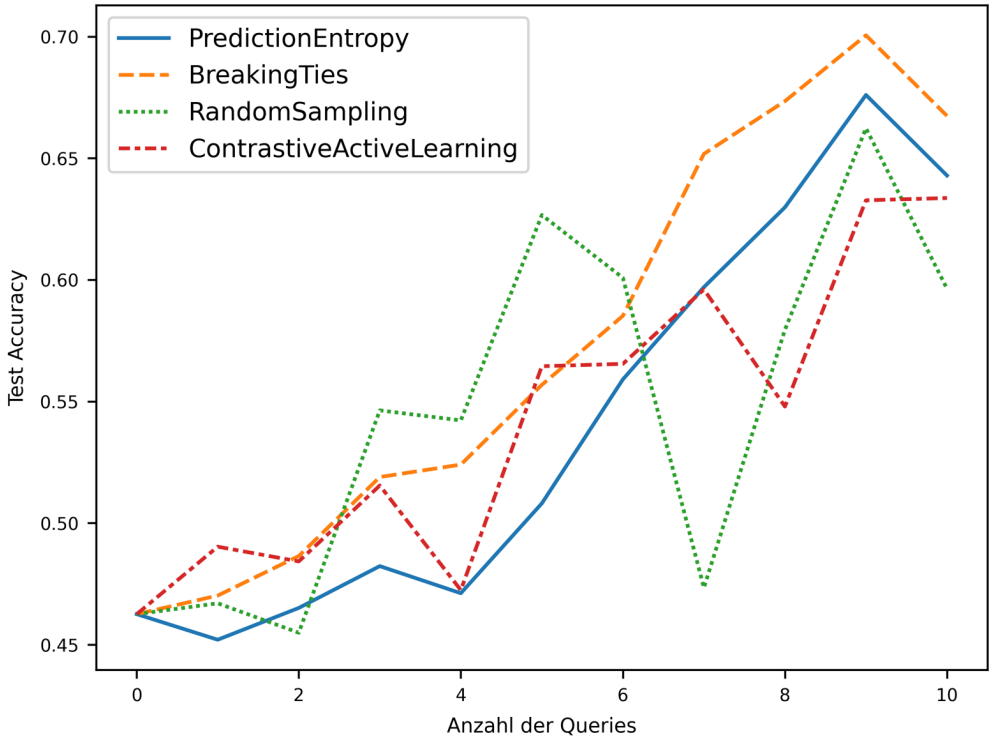

Abbildung 19: Übersicht: Accuracy aller Query Strategien pro Query (GPT-2)

Es gibt auf den ersten Blick keine Query Strategie, die konsistent eine bessere Accuracy als die anderen erreicht hat. Insbesondere beim Random Sampling fällt auf, dass es einen starken Anstieg zwischen Iteration 2 und 5 gab. Dieser Anstieg wurde jedoch von einer starken Senkung der Accuracy zwischen Iteration 5 und 7 gefolgt. Das Random Sampling war zwischen Query 3 und 6 die Query Strategie mit der höchsten Accuracy. Jedoch führte die zufallsbasierte Auswahl zu inkonsistenten Ergebnissen, was sich in den starken Schwankungen der Accuracy widerspiegelte. Sieht man daher vom Random Sampling ab, so lag die Lernkurve von Breaking Ties die meiste Zeit oberhalb der anderen Lerkurven. Bei Query 5 lag die Accuracy des Contrastive Active Learning nur kurzzeitig oberhalb der Accuracy von Breaking Ties. Die Prediction Entropy war ebenfalls konsistenter als das Random Sampling und lag durchgehend knapp unter der Accuracy von Breaking Ties. Interessanterweise erzielten fast alle Query Strategien bei Query 9 eine bessere Accuracy als nach der Ausführung von 10 Queries. Nach der Ausführung von 10 Queries erreichte Breaking Ties die höchste Accuracy.

Aus der Übersichtstabelle 9 geht hervor, dass GPT-2 in Kombination mit Breaking Ties die besten Ergebnisse lieferte. Die genannte Kombination erreichte einen F_1-Score

von $0{,}62$. GPT-2 erzielte hingegen bei einem Training auf dem gesamten Trainingsdatensatz sogar einen F_1-Score von $0{,}74$. Somit konnte hier im Active Learning nur knapp 83% der Güte des Modells erhalten werden.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,19	0,12	**0,47**	0,08	48
Sonstiges	0,00	0,00	0,00	0,00	68
Fahrradparken	0,04	**0,80**	0,79	0,18	139
Beschilderung	0,00	0,04	**0,10**	0,07	185
Ampeln	**0,56**	0,47	0,46	0,34	259
Hindernisse	0,26	**0,57**	0,51	0,48	385
Radwegqualität	0,53	0,66	0,56	**0,67**	618
Radverkehrsführung	0,75	**0,76**	0,74	0,75	1437
micro avg	0,53	**0,62**	0,60	0,57	3139

Tabelle 9: Vergleich der F_1-Scores aller Query Strategien (GPT-2)

Die Breaking Ties Strategy war dicht gefolgt von der Prediction Entropy, die einen F_1-Score von $0{,}60$ erreichen konnte. Das Random Sampling erreichte gerade mal einen F_1-Score von $0{,}53$ und Contrastive Active Learning erreichte einen F_1-Score von $0{,}57$. Betrachtet man die F_1-Scores der einzelnen Klassen, so fällt auf, dass Breaking Ties bei drei Klassen die besten Werte lieferte, während Prediction Entropy bei zwei Klassen die besten F_1-Scores erzielte. Random Sampling hatte den höchsten F_1-Score bei der Klasse *Ampeln* und Contrastive Active Learning hatte den höchsten F_1-Score bei der *Radwegqualität*. Alle Query Strategien hatten mit GPT-2 einen F_1-Score von 0 bei der Klasse *Sonstiges*.

Die Zeitmessungen haben ergeben, dass das Random Sampling hier mit einer Zeit von 19:49 Minuten am schnellsten abgeschlossen war.

Query Strategie	Zeitmessung
Random Sampling	19:49
Breaking Ties	24:48
Prediction Entropy	25:13
Contrastive Active Learning	26:05

Tabelle 10: Vergleich: Zeitmessung nach Query Strategien (GPT-2)

Contrastive Active Learning hat mit einer Zeitmessung von 26:05 Minuten mit Abstand am längsten gebraucht. Breaking Ties war nach 24:48 Minuten abgeschlossen und die Prediction Entropy benötigte mit 25:13 Minuten nur knapp 25 Sekunden länger.

5.3.4 Übersichtsseite - Primärklassen mit Einzelzuweisung

Aus Tabelle 11 geht hervor, dass DistilBERT in Kombination mit einer Prediction Entropy den höchsten F_1-Score von $0{,}72$ erzielen konnte. Insgesamt konnte DistilBERT bei drei der vier Query Strategien die besten F_1-Scores erzielen.

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	**0,66**	0,70	0,69	0,65
DistilBERT	0,61	**0,71**	**0,72**	**0,67**
GPT-2	0,53	0,62	0,60	0,57

Tabelle 11: Vergleich der F_1-Scores aller Experimente (Primärklassen mit Einzelzuweisung)

DistilBERT konnte das BERT Modell äußerst knapp bei der Anwendung von Breaking Ties, Prediction Entropy und Contrastive Active Learning übertreffen. Beim Random Sampling führte BERT deutlich mit einem F_1-Score von $0{,}66$, DistilBERT erreichte hier nur einen F_1-Score von $0{,}61$. GPT-2 hingegen erzielte deutlich geringere F_1-Scores. Der höchste mit GPT-2 erzielte F_1-Score lag nur bei $0{,}62$ (Breaking Ties), was selbst vom Random Sampling mit BERT übertroffen werden konnte.

Aus Tabelle 12 geht hervor, dass die Active Learning-Verfahren mit DistilBERT tendenziell am schnellsten abgeschlossen waren.

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	06:57	09:11	08:58	09:42
DistilBERT	**04:41**	**05:44**	**05:38**	**06:43**
GPT-2	19:49	24:48	25:13	26:05

Tabelle 12: Vergleich der Zeitmessungen aller Experimente (Primärklassen mit Einzelzuweisung)

Das Experiment mit dem besten F_1-Score (DistilBERT + Prediction Entropy) hatte hier nur eine Laufzeit von 05:38 Minuten. Dies ist kürzer als die Laufzeit von BERT mit der schnellsten Query Strategie, dem Random Sampling. GPT-2 hat im Gegensatz dazu deutlich länger gebraucht als BERT oder DistilBERT. Es scheint aufgrund der geringen F_1-Scores und hohen Laufzeiten nicht besonders gut für das hier beschrieben Active Learning-Verfahren geeignet zu sein. So brauchte GPT-2 im Vergleich zu DistilBERT beispielsweise fast 5-mal länger für die Durchführung des Active Learning-Verfahrens mit einer Prediction Entropy und erzielte einen geringeren F_1-Score.

5.4 Primärklassen mit Mehrfachzuweisungen

5.4.1 BERT - Primärklassen mit Mehrfachzuweisung

Die Klassifikation der Primärklassen mit Mehrfachzuweisung mit BERT ergab, dass die Prediction Entropy und Breaking Ties äußerst gute Ergebnisse erzielten (siehe Abbildung 20).

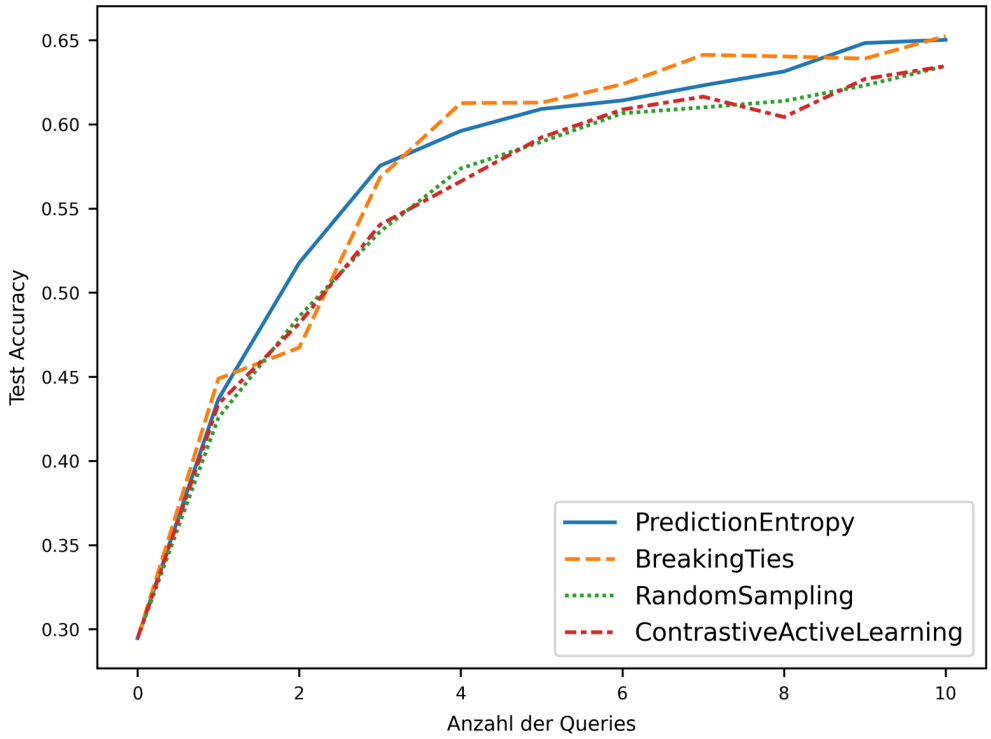

Abbildung 20: Übersicht: Accuracy aller Query Strategien pro Query (BERT)

Zu Beginn lagen alle Lernkurven realtiv nah beieinander, jedoch teilten sie sich ab Iteration 3 deutlich auf. Dabei fällt auf, dass die Accuracies des Random Sampling und Contrastive Active Learning deutlich unterhalb der Werte von Prediction Entropy und Breaking Ties lagen. Die Lernkurven von Breaking Ties und Prediction Entropy erreichten bereits nach Iteration 4 schon eine Accuracy von über $0{,}60$. Außerdem ist anzumerken, dass die Breaking Ties Strategie größeren Schwankungen unterlag, während die Prediction Entropy weniger von solchen Schwankungen betroffen war. Dementsprechend war hier je nach Iteration entweder Breaking Ties oder Prediction Entropy die bessere Query Strategie.

Die Auswertung der F_1-Scores ergab, dass es einen Gleichstand zwischen Breaking Ties und der Prediction Entropy gab (siehe Tabelle 13). Beide F_1-Scores lagen insgesamt bei $0{,}74$. Das Contrastive Active Learning lag mit $0{,}73$ knapp dahinter und das Random Sampling erreichte einen F_1-Score von $0{,}72$.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,76	**0,85**	0,84	0,84	64
Sonstiges	0,26	0,25	**0,35**	0,29	137
Fahrradparken	**0,87**	0,86	**0,87**	0,86	147
Beschilderung	0,48	**0,53**	0,47	0,52	229
Ampeln	0,75	0,75	**0,78**	0,77	287
Hindernisse	0,65	**0,69**	0,66	0,65	442
Radwegqualität	0,73	**0,74**	0,72	0,69	708
Radverkehrsführung	0,78	0,79	**0,80**	0,79	1490
micro avg	0,72	**0,74**	**0,74**	0,73	3504

Tabelle 13: Vergleich der F_1-Scores aller Query Strategien (BERT)

BERT konnte mit einem Training auf dem gesamten Trainingsdatensatz einen F_1-Score von $0{,}76$ erzielen. Also sind auch hier die Active Learning Ergebnisse vielversprechend, da bis zu 97% der Güte des Modells im Active Learning Szenario erhalten werden konnten. Bei der Klasse *Fahrradparken* erreichte die Prediction Entropy mit $0{,}87$ einen äußerst hohen Wert für den F_1-Score. Auch bei der äußerst schwer zu klassifizierenden Klasse *Sonstiges* konnte die Prediction Entropy einen vergleichsweise hohen Wert von $0{,}35$ erreichen. Die am stärksten repräsentierte Klasse *Radverkehrsführung* konnte ebenfalls von der Prediction Entropy mit einem Höchstwert von $0{,}80$ für den F_1-Score klassifiziert werden.

Die Zeitmessungen der Klassifikation der Primärklassen mit Mehrfachzuweisungen hat mit BERT mehr als dreimal so lange gedauert wie die Klassifikation der Primärklassen mit Einzelzuweisungen (siehe Tabelle 14). Allerdings konnten bei der Klassifikation auch etwas höhere F_1-Scores erzielt werden.

Query Strategie	Zeitmessung
Random Sampling	23:09
Breaking Ties	34:25
Prediction Entropy	34:52
Contrastive Active Learning	34:28

Tabelle 14: Vergleich: Zeitmessung nach Query Strategien (BERT)

Das Random Sampling war dabei mit knapp 23 Minuten wieder die schnellste Query Strategie und war damit etwa 30% schneller als die anderen Query Strategien. Breaking Ties, Prediction Entropy und Contrastive Learning brauchten hier jeweils circa 34 Minuten. Allerdings sollte auch in Betracht gezogen werden, dass beispielsweise Breaking Ties schon nach 4 Iterationen eine sehr hohe Accuracy erreichen konnte (siehe Abbildung 20).

5.4.2 DistilBERT - Primärklassen mit Mehrfachzuweisung

Die Klassifikation der Primärklassen mit Mehrfachzuweisung ergab, dass Contrastive Active Learning und Breaking Ties in Kombination mit DistilBERT die besten Ergebnisse erzielten (siehe Abbildung 21).

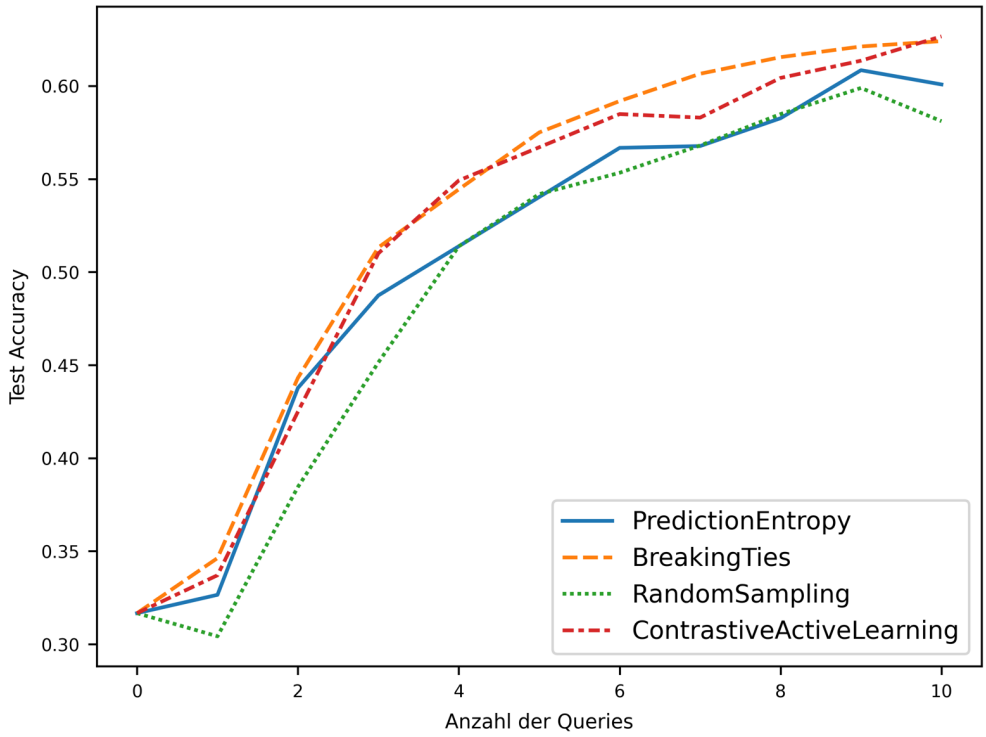

Abbildung 21: Übersicht: Accuracy aller Query Strategien pro Query (DistilBERT)

Die Lernkurven dieser beiden Query Strategien befanden sich fast durchgehend oberhalb der Lernkurven von Prediction Entropy und Random Sampling. Das Random Sampling war hier eindeutig das am wenigsten erfolgreiche Verfahren, was sich im Verlauf von dessen Lernkurve widerspiegelte. Die Prediction Entropy erzielte dabei in einigen Iterationen eine etwas bessere Accuracy als das Random Sampling. Insgesamt lagen die Zwischenergebnisse der Prediction Entropy und des Random Sampling jedoch ziemlich nah beieinander.

Die Untersuchungen der F_1-Scores in diesem Setting ergaben, dass DistilBERT in Kombination mit Contrastive Active Learning den höchsten F_1-Score von $0{,}73$ erreichen konnte. Die F_1-Scores aller Query Strategien lagen zwischen $0{,}69$ und $0{,}73$ und das Random Sampling erzielte hier mit $0{,}69$ den niedrigsten F_1-Score. Bei einem Training auf dem gesamten Trainingsdatensatz erzielte DistilBERT einen F_1-Score von $0{,}76$. Somit konnten im Active Learning Setting bis zu 96% der Güte von DistilBERT erhalten werden.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,80	**0,87**	0,80	0,81	64
Sonstiges	0,27	0,22	**0,37**	0,17	137
Fahrradparken	0,86	**0,87**	0,85	**0,87**	147
Beschilderung	**0,54**	0,45	0,46	0,50	229
Ampeln	0,77	0,77	**0,79**	0,78	287
Hindernisse	0,61	0,64	0,61	**0,66**	442
Radwegqualität	0,67	**0,70**	0,68	0,69	708
Radverkehrsführung	0,74	**0,79**	0,77	**0,79**	1490
micro avg	0,69	0,72	0,71	**0,73**	3504

Tabelle 15: Vergleich der F_1-Scores aller Query Strategien (DistilBERT)

Insgesamt ging das Contrastive Active Learning nur bei der Klasse *Hindernisse* mit einem F_1-Score von $0{,}66$ als klarer Sieger hervor. Jedoch erzielte es insgesamt einen sehr hohen F_1-Score, da es bei fast allen Klassen relativ hohe Werte für den F_1-Score hatte. Bei der am stärksten repräsentierten Klasse *Radverkehrsführung* erreichten sowohl Breaking Ties als auch Contrastive Active Learning einen sehr hohen F_1-Score von $0{,}79$. Es fällt außerdem auf, dass die Prediction Entropy hier bei der Klasse *Sonstiges* einen vergleichsweise hohen F_1-Score von $0{,}37$ erreicht hat. Die Klasse *Sonstiges* galt in den vorherigen Experimenten als eine der am schwersten zu klassifizierenden Klassen und erreichte oft nur einen F_1-Score von 0.

DistilBERT brauchte für die Klassifikation der Primärklassen mit Mehrfachzuweisungen etwa dreimal so lange wie für die Klassifikation der Primärklassen mit Einzelzuweisungen (siehe Tabelle 16). Dabei konnten je nach Query Strategie vergleichbare oder minimal höhere F_1-Scores erzielt werden.

Query Strategie	Zeitmessung
Random Sampling	12:20
Breaking Ties	18:21
Prediction Entropy	19:00
Contrastive Active Learning	18:14

Tabelle 16: Vergleich: Zeitmessung nach Query Strategien (DistilBERT)

Das Random Sampling war hier mit 12:20 Minuten wieder die schnellste Strategie, erreichte jedoch auch den geringsten F_1-Score. Die drei Query Strategien Breaking Ties, Prediction Entropy und Contrastive Active Learning brauchten zwischen 18 und 19 Minuten. Auch hier sollte wieder angemerkt werden, dass das Random Sampling insbesondere bei einer geringeren Anzahl an Iterationen schlechtere Ergebnisse lieferte als die anderen Query Strategien (siehe Abbildung 21).

5.4.3 GPT-2 - Primärklassen mit Mehrfachzuweisung

Die Auswertungen der Klassifikation von Primärklassen mit Mehrfachzuweisung mit GPT-2 haben ergeben, dass das Random Sampling hier deutlich inkonsistenter als die anderen Query Strategien arbeitet (siehe Abbildung 22).

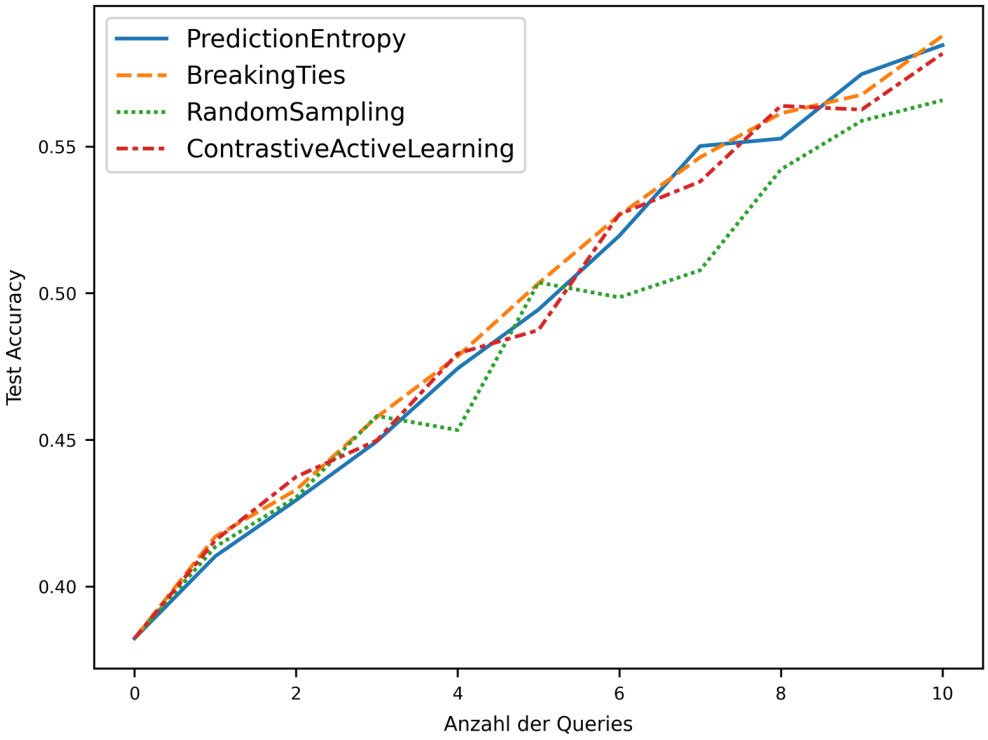

Abbildung 22: Übersicht: Accuracy aller Query Strategien pro Query (GPT-2)

Das Random Sampling ist dabei nicht nur größeren Schwankungen ausgesetzt, es wird auch deutlich von den anderen Query Strategien übertroffen. Die Lernkurven von Prediction Entropy, Breaking Ties und Contrastive Active Learning liegen während des gesamten Experiments sehr dicht beieinander. Bei jeder der zehn Iterationen erreicht jeweils eine dieser drei Query Strategien den Höchstwert für die Accuracy. Während der Anstieg der Accuracies beim Random Sampling in den letzten zwei Iterationen leicht abflacht, steigen die Accuracies der anderen Strategien auch dort noch stetig an. Der starke Anstieg der Accuracy am Ende des Experiments lässt vermuten, dass eine leichte Erhöhung der Iterationen hier zu einer deutlichen Verbesserung der Ergebnisse führen könnte.

Der höchste F_1-Score konnte mit einer Kombination aus GPT-2 und Contrastive Active Learning erreicht werden und lag bei $0{,}70$. Dabei erreichten alle Query Strategien insgesamt Werte zwischen $0{,}68$ und $0{,}70$ für den F_1-Score. Breaking Ties und Prediction Entropy erreichten jeweils einen F_1-Score von $0{,}69$ und das Random Sampling war mit $0{,}68$ die am wenigsten erfolgreiche Query Strategie. Bei einem Training auf dem

gesamten Trainingsdatensatz erzielte GPT-2 einen F_1-Score von $0{,}76$. Demnach konnten hier bis zu 92% der Güte von GPT-2 mithilfe von Active Learning-Verfahren erreicht werden.

	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning	Support
Beleuchtung	0,54	0,69	**0,72**	0,41	64
Sonstiges	0,03	0,06	**0,10**	0,03	137
Fahrradparken	0,80	**0,81**	0,79	0,78	147
Beschilderung	0,39	0,42	0,42	**0,50**	229
Ampeln	0,70	**0,72**	**0,72**	**0,72**	287
Hindernisse	0,59	**0,65**	0,63	0,60	442
Radwegqualität	0,66	0,61	0,66	**0,67**	708
Radverkehrsführung	0,76	0,77	0,76	**0,78**	1490
micro avg	0,68	0,69	0,69	**0,70**	3504

Tabelle 17: Vergleich der F_1-Scores aller Query Strategien (GPT-2)

Das Contrastive Active Learning konnte insbesondere bei der Klasse *Beschilderung* einen deutlich höheren F_1-Score als die anderen Query Strategien erreichen. So lag der Wert für das Contrastive Active Learning bei $0{,}50$, während die anderen Query Strategien höchstens einen Wert von $0{,}42$ erzielen konnten. Bei der Klasse *Ampeln* gab es einen Gleichstand zwischen Breaking Ties, Prediction Entropy und Contrastive Active Learning. Alle drei Query Strategien erreichten hier einen F_1-Score von $0{,}72$. Das Random Sampling konnte bei keiner der acht Primärklassen die besten Ergebnisse erzielen.

Bei der Klassifikation der Primärklassen mit Mehrfachzuweisungen mit GPT-2 war wieder das Random Sampling mit 19:13 Minuten die schnellste Query Strategie (siehe Tabelle 18). Es fällt auf, dass sich die erforderlichen Zeiten im Vergleich zur Klassifikation der Primärklassen mit Einzelzuweisungen kaum verändert haben. Allerdings gab es hier eine deutliche Verbesserung der F_1-Scores.

Query Strategie	Zeitmessung
Random Sampling	19:13
Breaking Ties	24:23
Prediction Entropy	24:22
Contrastive Active Learning	25:22

Tabelle 18: Vergleich: Zeitmessung nach Query Strategien (GPT-2)

Auch hier sollte angemerkt werden, dass das Random Sampling während des Trainings sehr starken Schwankungen unterlag (siehe Tabelle 22). Die anderen drei Query Strategien brauchten insgesamt nur circa 5 bis 6 Minuten länger als das Random Sampling, waren jedoch konsistenter und lieferten nach zehn Iterationen etwas bessere Ergebnisse.

5.4.4 Übersichtsseite - Primärklassen mit Mehrfachzuweisung

Die Gegenüberstellung der verschiedenen Klassifikationsalgorithmen hat gezeigt, dass BERT insgesamt die besten Ergebnisse erzielen konnte (siehe Tabelle 19). Dabei gab es in diesem Setting einen Gleichstand zwischen BERT und DistilBERT für das Contrastive Active Learning. Beide Classifier erreichten hier einen F_1-Score von $0{,}73$.

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	**0,72**	**0,74**	**0,74**	**0,73**
DistilBERT	0,69	0,72	0,71	**0,73**
GPT-2	0,68	0,69	0,69	0,70

Tabelle 19: Vergleich der F_1-Scores aller Experimente (Primärklassen mit Mehrfachzuweisung)

Den besten F_1-Score von $0{,}74$ erreichte BERT sowohl mit Breaking Ties als auch mit der Prediction Entropy. GPT-2 konnte in diesem Setting deutlich bessere Ergebnisse erreichen als bei der Klassifikation von Primärklassen mit Einzelzuweisung. So erreichte GPT-2 in Kombination mit Contrastive Active Learning hier einen realtiv hohen F_1-Score von $0{,}70$, während GPT-2 zuvor nur Höchstwert von $0{,}62$ erreichen konnte. Interessanterweise wurden die Höchstwerte der anderen beiden Klassifikationsalgorithmen mithilfe von Contrastive Active Learning erreicht. Insgesamt fiel auf, dass das Random Sampling unabhängig vom Klassifikationsalgorithmus die geringsten F_1-Scores erzielte. Dies lässt darauf schließen, dass die gezielte Auswahl von Datenpunkten mithilfe der Query Strategien zu einer Verbesserung der Ergebnisse beigetragen hat.

Die Zeitmessungen ergaben auch hier, dass die Experimente mit DistilBERT am schnellsten abgeschlossen waren (siehe Tabelle 20).

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	23:09	34:25	34:52	34:28
DistilBERT	**12:20**	**18:21**	**19:00**	**18:14**
GPT-2	19:13	24:23	24:22	25:22

Tabelle 20: Vergleich der Zeitmessungen aller Experimente (Primärklassen mit Mehrfachzuweisung)

DistilBERT in Kombination mit Contrastive Active Learning brauchte beispielsweise nur 18:14 Minuten und erzielte mit einem F_1-Score von $0{,}73$ das zweitbeste Ergebnis in der Klassifikation.

5.5 Sekundärklassen mit Mehrfachzuweisungen

5.5.1 BERT - Sekundärklassen mit Mehrfachzuweisung

Betrachtet man die Lernkurven der verschiedenen Query Strategien in Abbildung 23, so fällt auf, dass die Kurven wesentlich näher beieinander liegen als beim Setting der Primärklassen.

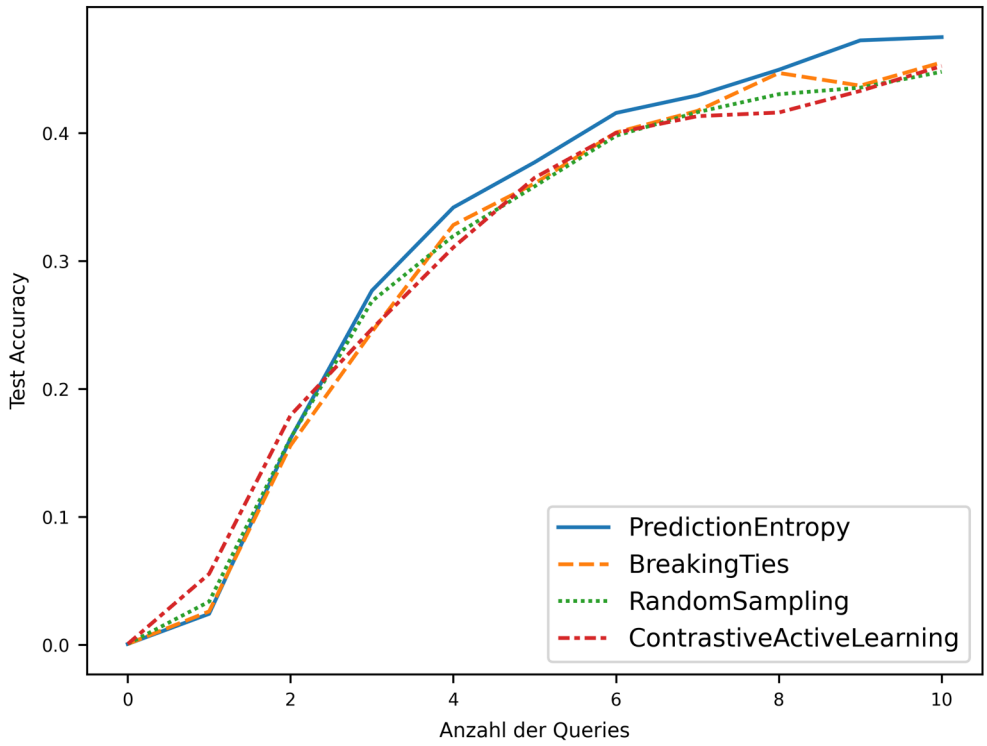

Abbildung 23: Übersicht: Accuracy aller Query Strategien pro Query (BERT)

Ab Iteration 3 liegt die Kurve der Prediction Entropy zwar knapp, jedoch konsistent oberhalb der anderen Lernkurven. Die anderen drei Query Strategien liegen jedoch fast durchgehend sehr dicht beieinander. In Iteration 8 teilt sich dieser Verlauf kurz auf, sodass die Kurve von Breaking Ties über der Kurve des Random Sampling und des Contrastive Active Learnings steht. Daraufhin führen die drei Kurven jedoch wieder zusammen.

Diese Ergebnissee spiegeln sich auch in Tabelle 21 wider. Die F_1-Scores aller Verfahren liegen zwischen $0{,}56$ und $0{,}58$ und sind somit sehr dicht beieinander. Den höchsten Wert von $0{,}58$ erzielte die Prediction Entropy. Bei einem Training auf dem gesamten Trainingsdatensatz erreichte BERT hier nur einen F_1-Score von $0{,}52$. Dies kann vermutlich darauf zurückgeführt werden, dass kein Hyperparameter-Tuning stattgefunden hat. Demnach kann es sein, dass die gewählten Hyperparameter besser für ein Active Learning-Verfahren geeignet waren als für ein Training des gesamten

Trainingsdatensatzes.
In Tabelle 21 lässt sich erkennen, dass Klassen mit schwacher Repräsentation sehr problematisch in der Klassifikation gewesen sind. So ist besonders auffällig, dass vier der sechs am schwächsten repräsentierten Klassen mit allen verwendeten Query Strategien einen F_1-Score von 0 hatten. Interessanterweise erzielten alle Klassen mit nur 38 oder mehr Repräsentanten F_1-Scores, die größer als 0 waren.

	R.S.	B.T.	P.E.	C.A.L.	Support
Falsche Beleuchtung	0,00	0,00	0,00	0,00	8
Ampel entfernen	**0,35**	0,25	0,00	0,00	13
Mängelmeldung	0,00	0,00	0,00	0,00	15
Ungeeignete Abstellanlagen	0,10	0,09	0,10	**0,24**	19
Auffahrt auf Radweg nur mit Umweg möglich	0,00	0,00	0,00	0,00	24
Sonstige Hinweise	0,00	0,00	0,00	0,00	32
Geschwindigkeitsbegrenzung	0,36	0,14	0,38	**0,47**	38
Radwegebenutzungspflicht überprüfen	0,65	**0,70**	**0,70**	0,67	45
Beleuchtung fehlt	0,72	0,66	0,78	**0,79**	56
Radweg beidseitig befahren	0,41	0,35	**0,45**	0,39	60
Wiederholt Schmutz oder Wasser auf Radweg	0,50	0,52	0,47	**0,57**	64
Radweg häufig blockiert	0,16	0,15	0,18	**0,26**	67
Übergänge mit zu großen Höhenunterschieden	0,67	0,69	**0,70**	0,66	72
Einbahnstraße für Radverkehr öffnen	0,70	**0,77**	0,73	0,73	74
Radwegweisung fehlt oder schlecht s.	0,40	0,45	**0,46**	0,41	84
Regelwidriges Verhalten	0,19	0,30	**0,37**	0,27	87
Ampel(-ergänzung) vorschlagen	0,35	0,44	**0,49**	0,45	92
Fahrradstrasse einrichten	0,25	**0,40**	0,35	0,36	93
Nicht ortsgebundene Vorschläge	0,31	**0,41**	0,34	0,32	104
Mangelnde Sichtbeziehungen	0,42	0,39	**0,44**	**0,44**	107
Keine oder zu wenig Abstellmöglichkeiten	0,85	0,85	**0,87**	**0,87**	130
Behinderung durch feste Gegenstände	0,65	0,64	0,64	**0,68**	131
Fahrbahnmarkierung Radweg fehlt, schlecht s.	0,54	0,50	**0,58**	0,56	147
Ampelschaltung ungünstig	0,74	**0,77**	0,76	0,73	184
Sichere Straßenquerung fehlt	0,42	**0,50**	0,48	0,46	207
Unklare Verkehrsführung für Radfahrende	0,37	**0,41**	0,37	0,25	255
Radweg permanent zugeparkt	0,70	0,74	**0,75**	**0,75**	257
Zu geringe Breite	**0,55**	0,49	0,51	0,53	302
Unebenheit Brüche oder Risse	**0,77**	0,76	0,75	**0,77**	329
Vorschlag für neuen Radweg	0,59	0,55	**0,60**	**0,60**	601
micro avg	0,56	0,56	**0,58**	0,57	3697

Tabelle 21: Vergleich der F_1-Scores aller Query Strategien (BERT)

Es sollte jedoch auch angemerkt werden, dass die starke Repräsentation einer Klasse nicht der einzige Faktor ist, der zu einem hohen F_1-Score beiträgt. Betrachtet man die am stärksten repräsentierte Klasse *Vorschlag für neuen Radweg* mit 601 Repräsentanten, so

fällt auf, dass sie höchstens einen F_1-Score von $0{,}60$ erreichte. Die in etwa halb so stark repräsentierte Klasse *Unebenheit Brüche oder Risse* hingegen erreichte F_1-Scores zwischen $0{,}75$ und $0{,}77$. Die Klasse *Beleuchtung fehlt* erzielte sogar trotz eines noch geringeren Supports von 56 einen F_1-Score von $0{,}79$.

5.5.2 DistilBERT - Sekundärklassen mit Mehrfachzuweisung

Die Lernkurven der verschiedenen Query Strategien für DistilBERT liegen wie auch bei BERT sehr dicht beieinander (siehe Abbildung 24).

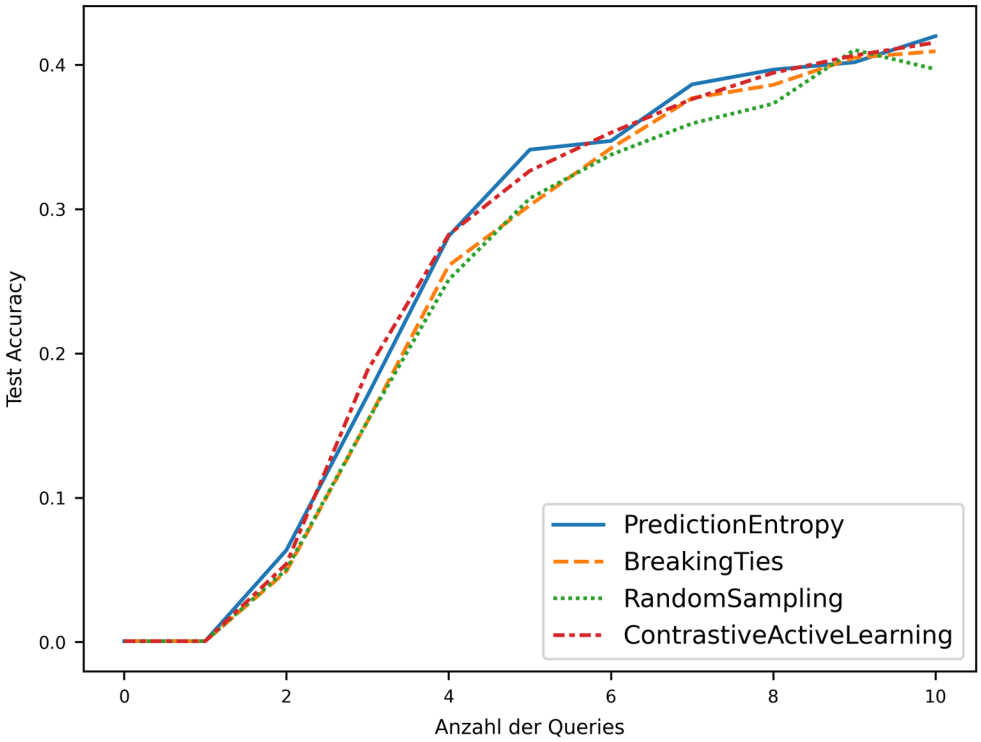

Abbildung 24: Übersicht: Accuracy aller Query Strategien pro Query (DistilBERT)

Im Gegensatz zu BERT ist jedoch die Kurve der Prediction Entropy nicht bei jeder Query konsistent oberhalb der anderen Kurven. Bei Query 9 ist die Accuracy der Prediction Entropy sogar geringer als die der anderen Query Strategien. Jedoch liegt die Accuracy-Kurve der Prediction Entropy nach 10 Iterationen oberhalb der anderen Kurven. Zwischen Query 4 bis 5 und Query 6 bis 8 gehen alle Kurven etwas weiter auseinander. In diesen Bereichen führt größtenteils die Prediction Entropy mit der höchsten Accuracy.

Aus Tabelle 22 geht hervor, dass die Prediction Entropy mit $0{,}55$ den höchsten F_1-Score erreichte. Es sollte jedoch angemerkt werden, dass die F_1-Scores der anderen Query Strategien mit Werten zwischen $0{,}53$ bis $0{,}54$ nur marginal darunter lagen.

DistilBERT erreichte beim Training auf dem gesamten Trainingsdatensatz sogar nur einen F_1-Score von $0{,}46$. Dieser äußerst geringe Wert lässt sich vermutlich auch hier auf das fehlende Hyperparameter-Tuning zurückführen.

	R.S.	B.T.	P.E.	C.A.L.	Support
Falsche Beleuchtung	0,00	0,00	0,00	0,00	8
Ampel entfernen	**0,38**	0,27	0,13	0,00	13
Mängelmeldung	0,00	0,00	0,00	0,00	15
Ungeeignete Abstellanlagen	0,10	0,17	0,16	**0,28**	19
Auffahrt auf Radweg nur mit Umweg möglich	0,00	0,00	0,00	0,00	24
Sonstige Hinweise	0,00	0,00	0,00	**0,11**	32
Geschwindigkeitsbegrenzung	0,42	0,49	**0,53**	0,43	38
Radwegebenutzungspflicht überprüfen	0,63	**0,66**	0,64	0,60	45
Beleuchtung fehlt	**0,74**	0,69	0,73	**0,74**	56
Radweg beidseitig befahren	0,10	**0,36**	0,26	0,32	60
Wiederholt Schmutz oder Wasser auf Radweg	0,32	0,38	**0,46**	**0,46**	64
Radweg häufig blockiert	0,05	0,08	0,05	**0,15**	67
Übergänge mit zu großen Höhenunterschieden	**0,67**	0,66	0,61	0,63	72
Einbahnstraße für Radverkehr öffnen	0,68	0,72	**0,73**	0,69	74
Radwegweisung fehlt oder schlecht s.	0,44	**0,45**	0,40	0,43	84
Regelwidriges Verhalten	0,20	**0,29**	0,21	0,23	87
Ampel(-ergänzung) vorschlagen	0,24	**0,36**	0,33	0,35	92
Fahrradstrasse einrichten	0,27	0,44	0,51	**0,52**	93
Nicht ortsgebundene Vorschläge	0,28	**0,33**	0,29	0,26	104
Mangelnde Sichtbeziehungen	0,29	0,37	0,34	**0,39**	107
Keine oder zu wenig Abstellmöglichkeiten	0,82	0,82	**0,85**	0,84	130
Behinderung durch feste Gegenstände	0,61	0,60	0,64	**0,66**	131
Fahrbahnmarkierung Radweg fehlt, schlecht s.	0,49	0,46	**0,50**	**0,50**	147
Ampelschaltung ungünstig	0,70	0,73	0,73	**0,74**	184
Sichere Straßenquerung fehlt	0,44	**0,47**	0,39	0,45	207
Unklare Verkehrsführung für Radfahrende	0,36	**0,38**	0,28	0,27	255
Radweg permanent zugeparkt	0,71	0,71	**0,72**	**0,72**	257
Zu geringe Breite	**0,53**	0,51	0,51	0,52	302
Unebenheit Brüche oder Risse	0,72	0,70	**0,76**	0,72	329
Vorschlag für neuen Radweg	0,54	0,56	**0,58**	0,54	601
micro avg	0,53	0,54	**0,55**	0,54	3697

Tabelle 22: Vergleich der F_1-Scores aller Query Strategien (DistilBERT)

Bei Betrachtung von Tabelle 22 fällt auf, dass schlecht repräsentierte Klassen tendenziell einen geringeren F_1-Score aufwiesen. Das Contrastive Active Learning schien bei diesen Klassen relativ gute Ergebnisse zu liefern. So erzielte es bei der Klasse *Ungeeignete Abstellanlagen*, die nur 19 Repräsentanten hat, mit $0{,}28$ den höchsten F_1-Score. Zusätzlich konnte das Contrastive Active Learning die Klasse *Sonstige Hinweise* mit einem F_1-Score von $0{,}11$ klassifizieren. Mit den anderen Query Strategien konnte für diese Klasse nur

ein von F_1-Score 0 erreicht werden. Bei den zwei am stärksten repräsentierten Klassen *Vorschlag für neuen Radweg* und *Unebenheit Brüche oder Risse* erzielte jedoch wieder die Prediction Entropy die besten Ergebnisse.

5.5.3 GPT-2 - Sekundärklassen mit Mehrfachzuweisung

Abbildung 25 zeigt, dass die Lernkurven der Query Strategien in Kombination mit GPT-2 zu Beginn sehr dicht beieinander liegen und sich dann aufteilen.

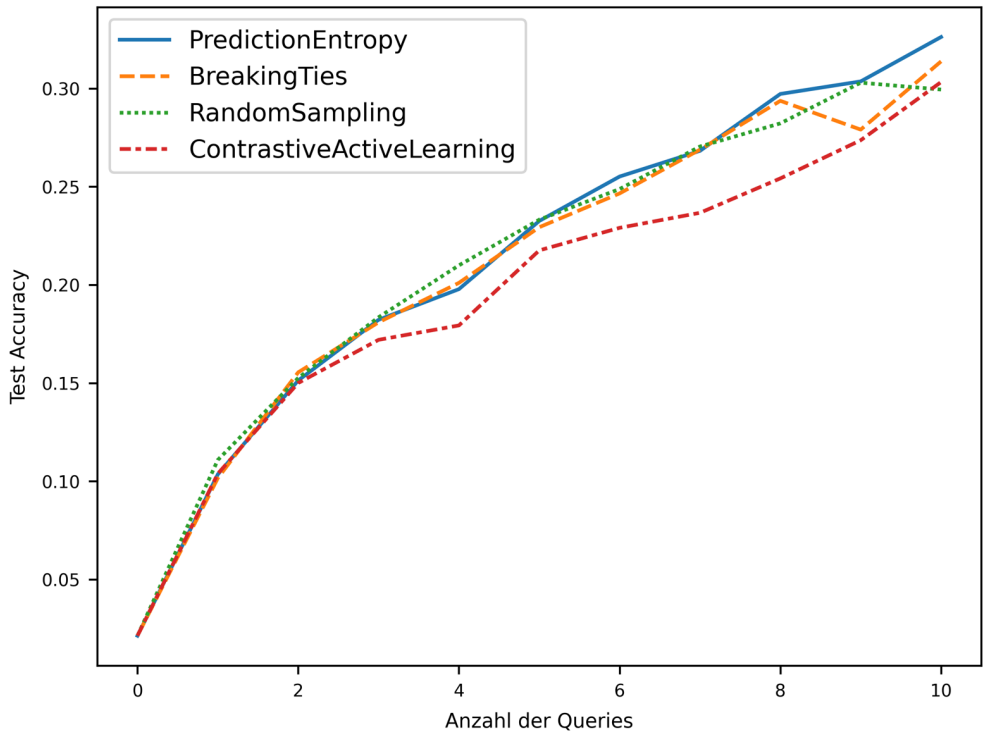

Abbildung 25: Übersicht: Accuracy aller Query Strategien pro Query (GPT-2)

Dabei fällt auf, dass auch hier die Accuracy der Prediction Entropy in den späten Iterationen 8 bis 10 oberhalb der Accuracy aller anderen Query Strategien liegt. Die Accuracy-Kurve des Contrastive Active Learning liegt deutlich unterhalb den Kurven der anderen Query Strategien. Die Kurven der anderen drei Verfahren liegen wesentlich dichter aneinander und es fällt auf, dass sowohl Breaking Ties als auch Random Sampling stärkeren Schwankungen unterliegen als die Prediction Entropy.

Aus Tabelle 23 ergibt sich, dass die Prediction Entropy mit einem Wert von $0{,}46$ in diesem Setting den höchsten F_1-Score erzielte. Auch hier liegen die F_1-Scores sehr dicht beieinander. Das Random Sampling hatte mit einem F_1-Score von $0{,}43$ den geringsten Wert. Beim Training mit dem gesamten Trainingsdatensatz konnte GPT-2 einen F_1-Score von $0{,}54$ erreichen. Somit blieben beim Einsatz von Active Learning-Verfahren und einer

reduzierten Trainingsmenge bis zu 85% der Güte des Modells erhalten.
Auch bei GPT-2 ergaben sich viele F_1-Scores von 0, wenn der Support der Klassen sehr gering war. Interessanterweise erreichte Contrastive Active Learning für die Klasse *Ampel entfernen* trotz des schwachen Supports von 13 einen F_1-Score von $0{,}25$. Die am stärksten repräsentierte Klasse *Vorschlag für neuen Radweg* konnte am besten mit Breaking Ties (F_1-Score von $0{,}54$) klassifiziert werden. Insgesamt sind die Werte der F_1-Scores mit dem Einsatz von GPT-2 deutlich geringer als mit den anderen Klassifikationsalgorithmen ausgefallen.

	R.S.	B.T.	P.E.	C.A.L.	Support
Falsche Beleuchtung	0,00	0,00	0,00	0,00	8
Ampel entfernen	0,14	0,00	0,00	**0,25**	13
Mängelmeldung	0,00	0,00	0,00	0,00	15
Ungeeignete Abstellanlagen	0,00	0,00	0,00	0,00	19
Auffahrt auf Radweg nur mit Umweg moeglich	0,00	0,00	0,00	0,00	24
Sonstige Hinweise	0,00	0,00	0,00	0,00	32
Geschwindigkeitsbegrenzung	0,10	0,23	**0,26**	0,14	38
Radwegebenutzungspflicht ueberpruefen	0,58	0,48	**0,59**	0,53	45
Beleuchtung fehlt	0,49	**0,50**	0,47	0,39	56
Radweg beidseitig befahren	0,03	**0,09**	**0,09**	0,06	60
Wiederholt Schmutz oder Wasser auf Radweg	0,19	**0,24**	0,12	0,14	64
Radweg häufig blockiert	0,00	0,03	**0,06**	**0,06**	67
Übergaenge mit zu großen Höhenunterschieden	**0,57**	0,53	0,51	0,40	72
Einbahnstraße für Radverkehr öffnen	0,72	0,66	**0,74**	0,66	74
Radwegweisung fehlt oder schlecht sichtbar	0,21	0,31	0,30	**0,32**	84
Regelwidriges Verhalten	0,04	0,13	0,14	**0,28**	87
Ampel(-ergänzung) vorschlagen	0,14	0,14	**0,21**	0,15	92
Fahrradstrasse einrichten	0,10	0,28	**0,32**	0,17	93
Nicht ortsgebundene Vorschläge	0,12	**0,19**	0,14	0,12	104
Mangelnde Sichtbeziehungen	0,14	0,17	0,21	**0,28**	107
Keine oder zu wenig Abstellmöglichkeiten	**0,77**	0,73	0,76	0,75	130
Behinderung durch feste Gegenstände	0,44	0,48	**0,55**	0,52	131
Fahrbahnmarkierung Radweg fehlt, schlecht s.	0,29	0,36	**0,40**	0,36	147
Ampelschaltung ungünstig	0,67	0,65	0,67	**0,68**	184
Sichere Straßenquerung fehlt	0,21	**0,33**	0,32	0,30	207
Unklare Verkehrsführung für Radfahrende	0,18	**0,28**	0,19	**0,28**	255
Radweg permanent zugeparkt	0,69	0,67	**0,70**	0,67	257
Zu geringe Breite	0,40	0,38	0,40	**0,45**	302
Unebenheit Brüche oder Risse	**0,67**	0,62	0,60	0,60	329
Vorschlag für neuen Radweg	0,46	**0,54**	0,52	0,45	601
micro avg	0,43	0,45	**0,46**	0,44	3697

Tabelle 23: Vergleich der F_1-Scores aller Query Strategien (GPT-2)

5.5.4 Übersichtsseite - Sekundärklassen mit Mehrfachzuweisung

Bemessen am F_1-Score erzielte BERT bei der Klassifikation der Sekundärklassen mit Abstand die besten Ergebnisse aller Klassifikationsalgorithmen (siehe Tabelle 24). DistilBERT konnte allerdings im Vergleich zu BERT eine ähnliche Güte der Ergebnisse erreichen.

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	**0,56**	**0,56**	**0,58**	**0,57**
DistilBERT	0,53	0,54	0,55	0,54
GPT-2	0,43	0,45	0,46	0,44

Tabelle 24: Vergleich der F_1-Scores aller Experimente (Sekundärklassen mit Mehrfachzuweisung)

Der höchste F_1-Score von $0{,}58$ konnte mit einer Kombination aus BERT und Prediction Entropy erzielt werden. GPT-2 hingegen war der Classifier mit den niedrigsten F_1-Scores. Sie lagen nur zwischen $0{,}43$ und $0{,}46$. Die Prediction Entropy war bei jedem Classifier die Query Strategie mit den besten Ergebnissen. Allerdings sollte angemerkt werden, dass die Unterschiede zwischen den Ergebnissen der Query Strategien bei gleichbleibendem Classifier eher marginal waren. Dies könnte darauf zurückzuführen sein, dass bei dem Setting mit 30 Sekundärklassen fast jeder Datenpunkt einen sehr hohen Informationsgehalt für das Training hatte. Dementsprechend spielt der Einsatz von Query Strategien hier möglicherweise erst bei einer größeren Anzahl an Datenpunkten für das Training eine relevante Rolle.

Die Auswertung aller Zeitmessungen ergab, dass DistilBERT im Vergleich zu den anderen Classifiern am schnellsten war (siehe Tabelle 25). Dies macht DistilBERT zu einem sehr effizienten Classifier, da es trotz geringerer Laufzeit fast die gleichen F_1-Scores wie BERT erzielte (siehe Tabelle 24). Im Kontrast dazu hatte GPT-2 die längsten Laufzeiten und die geringsten F_1-Scores.

Classifier	Random Sampling	Breaking Ties	Prediction Entropy	Contrastive Active Learning
BERT	28:48	39:20	40:29	43:20
DistilBERT	**24:14**	**27:29**	**27:01**	**30:23**
GPT-2	39:35	46:30	45:55	48:26

Tabelle 25: Vergleich der Zeitmessungen aller Experimente (Sekundärklassen mit Mehrfachzuweisung)

6 Fazit

6.1 Zusammenfassung und Interpretation der Ergebnisse

Aus den Untersuchungen ging hervor, dass BERT und DistilBERT bei der Klassifikation der Textbeiträge in allen Settings die besten Ergebnisse erzielen konnten. GPT-2 konnte in keinem der Experimente die besten Ergebnisse liefern. Es konnte allerdings in einigen ausgewählten Settings gute Ergebnisse erzielen, wie zum Beispiel bei der Klassifikation der Primärklassen mit Mehrfachzuweisungen. Die Zeitmessungen haben ergeben, dass GPT-2 grundsätzlich länger als die anderen beiden Klassifikationsalgorithmen für das Training benötigt hat. Somit sind BERT und DistilBERT aufgrund der besseren Güte und kürzeren Laufzeiten in Active Learning Szenarien deutlich attraktiver. Die F_1-Scores von BERT und DistilBERT lagen in den meisten Experimenten sehr dicht beieinander. In einigen Fällen, beispielsweise bei den Primärklassen mit Einzelzuweisungen, konnte DistilBERT sogar minimal höhere F_1-Scores erreichen als BERT. Dies könnte darauf zurückgeführt werden, dass kein Hyperparameter-Tuning durchgeführt wurde und die hier gewählten Hyperparameter besser für DistilBERT als für BERT geeignet waren. Ein Hyperparameter-Tuning könnte also an dieser Stelle äußerst hilfreich sein, um eine deutlichere Differenzierung der Ergebnisse zu erhalten. Insgesamt konnten mit allen Klassifikationsalgorithmen hohe Anteile der Güte der Modellvorhersage bei gleichzeitiger Reduktion der Trainingsdatenmenge im Active Learning erhalten werden.

Bei der Klassifikation der Primärklassen mit Einzelzuweisungen lieferten die beiden Uncertainty-Based Strategien Breaking Ties und Prediction Entropy für jeden Classifier die besten F_1-Scores. Das Random Sampling unterlag in diesem Setting starken Schwankungen und erreichte grundsätzlich geringere F_1-Scores als die anderen Query Strategien. Das Contrastive Active Learning erzielte in diesem Setting deutlich schlechtere Ergebnisse als Breaking Ties und Prediction Entropy. Dem könnte zu Grunde liegen, dass das Contrastive Active Learning nicht optimal für dieses Setting geeignet war. Datenpunkte, die mehreren Klassen zugeordnet werden konnten, wiesen vermutlich eine erhöhte Heterogenität und somit auch eine höhere Kullback-Leibler Divergenz zu ihrer Nachbarschaft auf. Dies könnte dazu geführt haben, dass diese Datenpunkte mit hoher Wahrscheinlichkeit zu Contrastive Examples deklariert und für das Training ausgewählt wurden. Allerdings lieferten diese Datenpunkte in diesem Setting aufgrund der Einzelzuweisungen nicht immer den größten Mehrwert für das Training, sodass Contrastive Active Learning hier ungeeignet war. Bei den Mehrfachzuweisungen hingegen könnte gerade dies ein Vorteil gegenüber den anderen Query Strategien sein.

Bei der Klassifikation von Primärklassen mit Mehrfachzuweisungen konnten höhere F_1-Scores als bei der Klassifikation der Primärklassen mit Einzelzuweisungen erreicht werden. Dies kann darauf zurückgeführt werden, dass bei gleicher Trainingsdatenmenge mehr Informationen über die Klassen erhalten werden können, da ein Datenpunkt mehreren Klassen zugeordnet worden sein kann. Hinzu kommt, dass bei den Einzelzuweisungen Inkonsistenzen der Annotation einen zusätzlichen negativen Einfluss auf die Güte der Modellvorhersage haben können. Dem liegt zu Grunde, dass bei Textbeiträgen mit mehreren Themenkategorien eine Entscheidung vom Annotator getroffen werden

muss, welche die primäre Klasse für die Einzelzuweisung ist. Diese Entscheidung ist in einigen Fällen nicht trivial, sodass auch der Klassifikationsalgorithmus bei diesen Textbeiträgen schlechtere Ergebnisse bei der Klassifikation erzielt.
Im Einsatz mit BERT konnten auch hier Breaking Ties und Prediction Entropy die besten Ergebnisse erzielen. Bei den anderen beiden Klassifikationsalgorithmen war jedoch das Contrastive Active Learning die beste Query Strategie. Wie bereits zuvor angemerkt, scheint Contrastive Active Learning besser für das Setting mit Mehrfachzuweisungen geeignet zu sein. Trotz einiger Schwankungen konnte auch das Random Sampling hier vergleichsweise gute Ergebnisse erzielen.

Bei der Klassifikation der Sekundärklassen hingegen fielen die F_1-Scores deutlich geringer aus als bei den anderen beiden Settings. Dies liegt höchstwahrscheinlich an der hohen Anzahl von 30 Sekundärklassen, die zusätzlich sehr imbalanciert sind. In diesem Setting konnte die Prediction Entropy bei jedem Klassifikationsalgorithmus die besten Ergebnisse in Bezug auf den F_1-Score erreichen. Breaking Ties erzielte bei allen Klassifikationsalgorithmen etwas geringere F_1-Scores als die Prediction Entropy. Dies kann vermutlich darauf zurückgeführt werden, dass sich die Datenauswahl der Prediction Entropy auf knappe Entscheidungen der Klassifikation fokussiert. Bei den Sekundärklassen spielt jedoch die vollständige Repräsentation aller Klassen eine wichtigere Rolle als eine Erhöhung der Confidence bei knappen Entscheidungen. Daher konnte die Anwendung von Contrastive Active Learning hier teilweise höhere F_1-Scores erzielen als Breaking Ties. Das Random Sampling unterlag hier weniger starken Schwankungen als bei den anderen beiden Settings.
Generell lagen die Ergebnisse der unterschiedlichen Query Strategien sehr dicht beieinander. Dies kann dadurch begründet werden, dass eine sehr hohe Anzahl an Klassen trotz sehr geringer Trainingsdatenmenge vorhanden war. Somit hatten fast alle Datenpunkte aufgrund der geringen Trainingsdatenmenge einen sehr hohen Informationsgehalt. Deshalb hatte die Anwendung unterschiedlicher Query Strategien in diesem Setting einen schwächeren Einfluss auf die Güte der Modellvorhersagen.

6.2 Ausblick und weitere Forschung

Die im Rahmen dieser Arbeit durchgeführten Ergebnisse konnten aufzeigen, dass der Einsatz von Active Learning-Verfahren ein vielversprechender Ansatz für die Reduktion des Annotationsaufwandes bei Klassifikationsproblemen ist. Für die zukünftige Forschung in diesem Bereich gibt es jedoch einige wichtige Aufgaben, die zu einer weiteren Verbesserung der Ergebnisse beitragen können.

Bei der Klassifikation der Sekundärklassen übertrafen beispielsweise BERT und DistilBERT im Active Learning-Verfahren die Referenzwerte der F_1-Scores, welche sich aus einem Training des gesamten Trainingsdatensatzes ergaben. Dies könnte darauf zurückzuführen sein, dass hier kein Hyperparameter-Tuning durchgeführt wurde. Dementsprechend ist es möglich, dass die gewählten Hyperparameter der Klassifikationsalgorithmen besser für das Training in einem Active Learning-Szenario geeignet waren, als für das Training des gesamten Trainingsdatensatzes. Durch ein Hyperparameter-Tuning könnten also aussagekräftigere Referenzwerte für den

Vergleich der hier implementierten Active Learning-Verfahren geschaffen werden. Desweiteren kann das Hyperparameter-Tuning zu einer Verbesserung der Güte der Modellvorhersagen führen.
Die Ergebnisse der Klassifikation der Sekundärklassen ließ außerdem darauf schließen, dass eine Änderung der initialen Trainingsdatenmenge, der Samples pro Iteration und der Anzahl der Iterationen ein aussichtsreicher Ansatz wäre. Denn bei den Sekundärklassen ergaben sich aufgrund der hohen Anzahl der Klassen nur wenige Unterschiede zwischen den Query Strategien. Somit könnte eine Erhöhung der Trainingsdatenmenge bei den Sekundärklassen zu wesentlich differenzierteren Ergebnissen führen.
Ein weiterer, interessanter Ansatz zur Verbesserung von Active Learning-Verfahren könnte der Austausch von Query Strategien nach einer bestimmten Anzahl von Iterationen sein. Die Lernkurven der verschiedenen Query Strategien flachen meist nach mehreren Iterationen ab. Es wäre daher interessant zu untersuchen, ob man dieser Stagnation mit einem Austausch durch eine andere Query Strategie entgegenwirken kann. Dadurch könnte innerhalb von wenigen Iterationen eine höhere Güte der Modellvorhersage erreicht werden.

Literatur

[1] Jimmy Lei Ba, Jamie Ryan Kiros und Geoffrey E Hinton. „Layer normalization“. In: *arXiv preprint arXiv:1607.06450* (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho und Yoshua Bengio. „Neural machine translation by jointly learning to align and translate“. In: *arXiv preprint arXiv:1409.0473* (2014).

[3] Michael Bloodgood und K Vijay-Shanker. „A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping“. In: *arXiv preprint arXiv:1409.5165* (2014).

[4] Cristian Buciluǎ, Rich Caruana und Alexandru Niculescu-Mizil. „Model compression“. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. (2006), S. 535–541.

[5] Branden Chan, Stefan Schweter und Timo Möller. „German's next language model“. In: *arXiv preprint arXiv:2010.10906* (2020).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova. „Bert: Pre-training of deep bidirectional transformers for language understanding“. In: *arXiv preprint arXiv:1810.04805* (2018).

[7] *Environmental Sampling Design*. John Wiley und Sons, Ltd, (2007). Kap. 3, S. 45–68.

[8] Philip Gage. „A new algorithm for data compression“. In: *C Users Journal* 12.2 (1994), S. 23–38.

[9] Santiago González-Carvajal und Eduardo C Garrido-Merchán. „Comparing BERT against traditional machine learning text classification“. In: *arXiv preprint arXiv:2005.13012* (2020).

[10] Geoffrey Hinton, Oriol Vinyals und Jeff Dean. *Distilling the Knowledge in a Neural Network*. (2015).

[11] Alex Holub, Pietro Perona und Michael C Burl. „Entropy-based active learning for object recognition“. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. (2008), S. 1–8.

[12] David D. Lewis und William Gale. „A sequential algorithmfor training text classifiers“. In: *Proc. of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*. (1994), S. 3–12.

[13] Tong Luo, Kurt Kramer, Dmitry B Goldgof, Lawrence O Hall, Scott Samson, Andrew Remsen, Thomas Hopkins und David Cohn. „Active learning to recognize multiple types of plankton.“ In: *Journal of Machine Learning Research* 6.4 (2005).

[14] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng und Christopher Potts. „Learning Word Vectors for Sentiment Analysis“. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, (2011), S. 142–150.

[15] Katerina Margatina, Giorgos Vernikos, Loïc Barrault und Nikolaos Aletras. „Active learning by acquiring contrastive examples“. In: *arXiv preprint arXiv:2109.03764* (2021).

[16] Malte Ostendorff, Till Blume und Saskia Ostendorff. „Towards an open platform for legal information“. In: *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*. (2020), S. 385–388.

[17] Stephen Purpura, Claire Cardie und Jesse Simons. „Active learning for e-rulemaking: Public comment categorization“. In: (2008).

[18] James Pustejovsky und Amber Stubbs. *Natural Language Annotation for Machine Learning*. O'Reilly Media, Inc., (2012).

[19] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever et al. „Improving language understanding by generative pre-training“. In: (2018).

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever et al. „Language models are unsupervised multitask learners“. In: *OpenAI blog* 1.8 (2019), S. 9.

[21] Julia Romberg und Tobias Escher. „Automated Topic Categorisation of Citizens' Contributions: Reducing Manual Labelling Efforts Through Active Learning“. In: *International Conference on Electronic Government*. Springer. (2022), S. 369–385.

[22] Victor Sanh, Lysandre Debut, Julien Chaumond und Thomas Wolf. „DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter“. In: *arXiv preprint arXiv:1910.01108* (2019).

[23] Christopher Schröder, Lydia Müller, Andreas Niekler und Martin Potthast. *Small-Text: Active Learning for Text Classification in Python*. (2021). arXiv: 2107.10314 [cs.LG].

[24] Rico Sennrich, Barry Haddow und Alexandra Birch. „Neural machine translation of rare words with subword units“. In: *arXiv preprint arXiv:1508.07909* (2015).

[25] Burr Settles. „Active learning literature survey“. In: (2009).

[26] Julien Simon. *Large Language Models: A New Moore's Law?* https://huggingface.co/blog/large-language-models. Aufgerufen am: 07.08.2022. (2021).

[27] Peter Usherwood und Steven Smit. „Low-shot classification: a comparison of classical and deep transfer machine learning approaches“. In: *arXiv preprint arXiv:1907.07543* (2019).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser und Illia Polosukhin. „Attention is all you need“. In: *Advances in neural information processing systems* 30 (2017).

[29] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy und Samuel R Bowman. „GLUE: A multi-task benchmark and analysis platform for natural language understanding“. In: *arXiv preprint arXiv:1804.07461* (2018).

[30] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz et al. „Huggingface's transformers: State-of-the-art natural language processing“. In: *arXiv preprint arXiv:1910.03771* (2019).

[31] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey et al. „Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[32] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba und Sanja Fidler. „Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *Proceedings of the IEEE international conference on computer vision*. (2015), S. 19–27.

Abbildungsverzeichnis

Tabellenverzeichnis