Institut für Maschinelle Sprachverarbeitung

Universität Stuttgart
Pfaffenwaldring 5b
D–70569 Stuttgart

Bachelorarbeit

Analyse der Repräsentanz von BERT-basierten Class-Embeddings in unbalancierten Datensätzen mittels Active Learning.

Alp Mujko

Studiengang:	Data Science
Prüfer/in:	Prof. Dr. Sebastian Padó
Betreuer/in:	Lukas Wertz, M.Sc.
Beginn am:	1. Dezember 2022
Beendet am:	1. Juni 2023

Kurzfassung

Das Sprachmodell BERT (*Bidirectional Encoder Representations from Transformers*) ist ein neuronales Netzwerk, das für die Verarbeitung von Textdaten ausgelegt ist und aufgrund seiner Fähigkeit, sowohl die Vorwärts- als auch die Rückwärtsrichtung des Kontexts zu berücksichtigen, als sehr leistungsfähig bei Aufgaben im Bereich der natürlichen Sprachverarbeitung gilt.

In dieser Arbeit nutzen wir dessen Transformer-Architektur, um geeignete Dokumenten-Embeddings für Texte zu generieren. Aus diesen leiten wir für jede Klasse an Dokumenten ein Class-Embedding ab, das stellvertretend für die jeweilige Klasse steht.

Um die Repräsentanz der erzeugten Class-Embeddings zu untersuchen, verwenden wir diese in einem Active Learning Szenario, um neue unbekannte Dokumente zu labeln.

Wir stellen fest, dass die berechneten Class-Embeddings ihre Klasse sinngemäß repräsentieren und folglich den Embedding-Raum sinnvoll partitionieren. Darüber hinaus liefert der Abstand zwischen den Class-Embeddings Aufschluss über die semantische Ähnlichkeit zwischen den Klassen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

In der heutigen Zeit findet der Einsatz künstlicher Intelligenzen in zahlreichen Bereichen unseres Lebens statt. Zum Beispiel übernehmen sie im Autonomen Fahren die Kontrolle über Fahrzeuge oder analysieren medizinische Bildaufnahmen in der medizinischen Diagnostik, um beispielsweise Krebs frühzeitig zu erkennen. Darüber hinaus werden sie auch in der maschinellen Sprachverarbeitung eingesetzt, wo sie Aufgaben wie Spracherkennung, Übersetzung und Textgenerierung übernehmen. Ein aktuell bekanntes Beispiel aus diesem Gebiet ist der Chatbot *ChatGPT*, der das Sprachmodell *GPT* verwendet, um die Eingaben der Nutzer zu verarbeiten.

Neben diesem Sprachmodell hat in den letzten Jahren das von der Firma *Google* entwickelte Sprachmodell *BERT* (*Bidirectional Encoder Representations from Transformers*) immer mehr an Relevanz gewonnen. BERT ist in der Lage komplexe Muster in Sprache zu erkennen und ein Verständnis über die Struktur der Texte zu entwickeln, die es als Eingabe bekommt. Es hat in zahlreichen Aufgaben der natürlichen Sprachverarbeitung, wie zum Beispiel Sentiment Analysis, Named Entity Recognition und Textklassifikation, State-of-the-Art Ergebnisse erzielt.

Besonders in der heutigen Zeit hat die Textklassifikation eine immense Relevanz erlangt. Durch die exponentielle Zunahme digitaler Inhalte, wie beispielsweise in sozialen Medien, Blogs, Online-Nachrichten und Kundenrezensionen, ist die Fähigkeit, Texte automatisch und effizient zu klassifizieren, von entscheidender Bedeutung. Ein anschauliches Beispiel hierfür ist die automatische Filterung von Spam-E-Mail. Hierfür wird ein Klassifizierer mit einem gelabelten Datensatz aus Spam-E-Mails und legitimen E-Mails trainiert. Damit lernt der Klassifizierer anhand dieser Daten, Muster und Merkmale zu erkennen, die auf Spam hinweisen. Basierend auf diesem Wissen kann er neue E-Mails automatisch als ”Spam” oder ”Legitim” klassifizieren. Dieses Beispiel verdeutlicht, wie die Textklassifikation mithilfe von maschinellen Lernalgorithmen funktioniert. Zunächst wird ein Trainingsdatensatz erstellt, der Textbeispiele enthält, die bereits manuell klassifiziert wurden. Anhand dieser gelabelten Daten lernt der Algorithmus, Muster und Merkmale in den Texten zu erkennen und automatisch Kategorien oder Klassen zuzuweisen.

Es stellt sich somit die interessante Frage, wie sehr sich BERT eignet Sprachstrukturen zu erlernen und diese verwendet werden können, um Textklassifikation durchzuführen.

Da jede textuelle Eingabe numerische Vektoren innerhalb von BERTs Architektur erzeugt, kann man annehmen, dass diese Vektoren BERTs Verständnis über den Text widerspiegelt.

Numerische Vektoren, die die enthaltene Information aus einem Text repräsentieren, werden auch *Embeddings* genannt. Trainiert man BERT auf Daten mit unterschiedlichen Klassen, könnte man anhand der erzeugten Embeddings *Class-Embeddings* erzeugen, die stellvertretend für die jeweilige Klasse stehen.

Sollte es BERT gelingen, semantisch repräsentative Class-Embeddings zu erzeugen, würden sich damit viele neue Möglichkeiten im Bereich der natürlichen Sprachverarbeitung ergeben.

Daher werden wir uns in dieser Arbeit mit der Erzeugung von Class-Embeddings beschäftigen und untersuchen, bis zu welchem Grad sie den semantischen Inhalt ihrer jeweiligen Klasse repräsentieren können.

Damit wir dieser Frage nachgehen können, entscheiden wir uns für ein *Active Learning* Setting. Active Learning ist eine Methode des maschinellen Lernens, bei der das Modell mit einem kleinen initialen Datensatz trainiert wird, und dann Datenpunkte nach einem bestimmten Kriterium aussucht, die von einem Annotator annotiert werden. Die neuen, annotierten Daten werden anschließend dem Trainingsdatensatz hinzugefügt.

Mit diesem Setting sind wir in der Lage zu untersuchen, wie repräsentativ die Class-Embeddings sind, indem wir sie für die Datenselektion nutzen und untersuchen, wie optimal diese neuen Datenpunkte aussuchen. Außerdem können wir mit diesem Setting die Veränderung der Performance der Class-Embeddings mit wachsendem Trainingsdatensatz untersuchen.

1.1 Verwandte Arbeit

Die Wirksamkeit von Active Learning für die Textklassifizierung wurde in vielen Studien untersucht. Neben Ansätzen [GKBG16; TK01] für Textklassifizierungen, die klassische Machine Learning Algorithmen verwenden, gibt es auch Ansätze [AWH18; SN20] für Textklassifizierungen, die Deep Learning Algorithmen verwenden. Diese Arbeiten fokussieren sich jedoch hauptsächlich auf die Untersuchung der Leistungssteigerung der Textklassifizierung mittels Active Learning. Die untersuchten Deep Learning Architekturen sind überwiegend *Convolutional Neural Network*, *Recurrent Neural Networks* und *Long Short-Term Memory*.

Ähnlich zu dieser Arbeit gibt es Studien [EHG+20; PMM21; SN20; WMKB22], die sich mit Active Learning für BERT-basierte Textklassifizierung befassen. Im Gegensatz zu unserem Ansatz werden keine generierten Embeddings für die Datenauswahl im Active Learning Prozess verwendet. Das Sprachmodell BERT [DCLT18] wird hierbei nur für die Textklassifikation verwendet.

Frühere Ansätze zur Generierung von Embeddings verwendeten trainierte Vektormodelle für Wörter [CHU17; PSM14]. Jedoch hat sich der Fokus mittlerweile auf die Nutzung der kontextbezogenen eingebetteten Informationen in großen Transformator-Sprachmodellen wie BERT verschoben.

Bereits in [DCLT18] wurde geforscht, wie man Embeddings, die von inneren Schichten der Architektur von BERT generiert werden, verwenden kann, um *Named Entity Recognition* Tasks zu bearbeiten. Es wurde gezeigt, dass dieser Ansatz eine hohe Performance in diesem Gebiet erzielt.

In [WBMK22] setzt man im Active Learning Prozess auf Embeddings, die von *Sentence-BERT* [RG19] generiert werden. Die erstellten Embeddings werden genutzt, um Class-Embeddings für Textklassen zu erstellen. Die Auswahl der zu annotierenden Datenpunkte erfolgt durch die Auswahl des Datums, dessen Embedding am nächsten zum Class-Embedding ist, dessen Klasse in den Trainingsdaten am wenigsten vertreten ist. Damit soll erzielt werden, dass jede Klasse in den Trainingsdaten gleich oft vertreten ist.

Diese Arbeit verfolgt einen ähnlichen Ansatz. Allerdings trainieren wir den Textklassifizierer auf den Daten, bevor wir die Embeddings durch die inneren Transformerschichten der BERT-Instanz im Kern des Klassifizierers generieren und verwenden.

2 Wissenschaftlicher Hintergrund

2.1 Das BERT-Modell

Im Rahmen dieser Bachelorarbeit wird für das von der Firma Google entwickelte BERT-Modell (*Bidirectional Encoder Representations from Transformers*) [DCLT18] untersucht. BERT ist ein neuronales Netzwerk, das für die Verarbeitung von Textdaten ausgelegt ist, und daher wird es häufig zur Durchführung von Aufgaben im Bereich der natürlichen Sprachverarbeitung verwendet. BERT hat sich dabei als sehr leistungsfähig erwiesen, insbesondere bei der Beantwortung von Fragen auf der Grundlage von gegebenen Texten [YDL+18], als auch in der Klassifikation von Texten [SQXH19].

2.1.1 Die Architektur des BERT-Modells

Wie bereits erwähnt ist BERT ein neuronales Netzwerk, welches aus mehreren Schichten besteht. Diese Schichten werden wir für unsere Untersuchungen verwenden, um die Dokumenten-Embeddings zu generieren. Als Grundlage für das BERT-Modell fungiert das Transformer-Modell [VSP+17]. Daher ist es nötig, dass wir das Tranformer-Modell genauer betrachten.

Das Transformer-Modell

Das Transformer-Modell [VSP+17] ist eine neuronale Netzwerkarchitektur, welche für die Machine Translation (maschinelle Übersetzung) von Texten entworfen wurde. Dabei setzt das Modell auf den Einsatz von *Self-Attention*, um die Beziehung zwischen Eingabetoken und den Ausgabetoken zu modellieren. Self-Attention ist ein Konzept des Deep Learnings, das es den Neuronen eines Netzwerks ermöglicht auf verschiedene Teile der Eingabedaten gleichzeitig zu achten, und somit die Beziehung zwischen diesen zu berücksichtigen [GLT21].

Im Gegensatz dazu wurden in der Vergangenheit auch *Convolutional Neural Networks* und *Recurrent Neural Networks* für dieselbe Art von Aufgaben verwendet [BCB14]. Es wurde jedoch gezeigt, dass diese im Vergleich zu dem Transformer-Modell mehr Ressourcen benötigen und gleichzeitig eine niedrigere Performance aufweisen [VSP+17].

Vom Transformer-Modell zum BERT-Modell

Das BERT-Modell baut auf dem Transformer-Modell [VSP+17] auf. Jedoch wurden Veränderungen vorgenommen, die zur Verbesserung des Verständnisses von Texten führen. In [DCLT18] werden die wichtigsten Änderungen aufgezählt:

- **Bidirektionale Verarbeitung:** Im Vergleich zu herkömmlichen Transformer-Modellen, die nur die Vorwärtsrichtung des Textes berücksichtigen, verarbeitet BERT Text bidirektional, das heißt, es berücksichtigt sowohl die Vorwärts- als auch die Rückwärtsrichtung des Textes. Dies hat erhebliche Auswirkungen auf die Leistung von BERT, da es in der Lage ist, beide Kontextrichtungen zu berücksichtigen, um Text besser zu verstehen. Beispielsweise kann ein Wort in einer kontextuellen Richtung eine völlig andere Bedeutung haben als in einer anderen. Mit der bidirektionalen Verarbeitung ist BERT in der Lage, beide Kontextrichtungen zu berücksichtigen und genauere Vorhersagen zu treffen.
- **Masked-Language Modeling:** Das ”Masked-Language Modeling”-Verfahren wird während des Trainingsprozesses von BERT eingesetzt. In diesem Verfahren werden zufällig ausgewählte Wörter im Text ”maskiert” (als unbekannt angesehen) und das Modell wird aufgefordert diese ”maskierten” Wörter vorherzusagen. Das bedeutet, dass das Modell eine Vorhersage darüber trifft, welches Wort an der Stelle des ”maskierten” Wortes stehen sollte. Aufgrund der bidirektionalen Natur von BERT werden hierbei sowohl die Vor- als auch die Rückwärtsrichtung des Kontexts berücksichtigt. Dieses Verfahren trägt dazu bei, dass das Modell ein besseres Verständnis des Textes erlangt und in der Lage ist, den Kontext und die Bedeutung von Wörtern in Bezug auf ihre Umgebung besser zu verstehen. Es hilft auch, die allgemeine Sprachverständnis des Modells zu verbessern, da es dazu angeregt wird, beide Kontextrichtungen zu berücksichtigen, wenn es Vorhersagen trifft.
- **Pretraining:** Das Modell wird mit einer großen Menge an Textdaten (vor-)trainiert. Durch das Pretraining lernt das Modell, die Bedeutung von Wörtern in verschiedenen Kontexten zu verstehen und wie sie interagieren, um die Bedeutung des Textes zu formen. Das Pretraining ermöglicht es, dass BERT besser auf bestimmte NLP-Aufgaben abgestimmt (fine-tuned) werden kann, da es bereits über ein tiefes Textverständnis verfügt.

Transformer-Schichten von BERT (Hidden-Layers)

Die BERT Hauptarchitektur besteht aus mehreren Transformer-Schichten. Jede dieser Transformer-Schichten besteht aus mehreren Sub-Schichten, einschließlich Multi-Head Attention Schichten und Feed-Forward-Schichten. Kurzgefasst berechnet die Multi-Head Attention Schicht die Bedeutung verschiedener Teile des Eingabetextes mithilfe von

parallelen Attention-Mechanismen, und die anschließende Feed-Forward Schicht transformiert und modifiziert die berechneten Bedeutungen durch eine Fully-Connected-Layer für eine detaillierte Repräsentation der Bedeutung der Eingabe. Für eine detaillierte Beschreibung der einzelnen Schichten verweisen wir an dieser Stelle auf das Originalpaper [DCLT18].

Abhängig von der Wahl des Modells hat BERT insgesamt 12 ($\text{BERT}_{\text{BASE}}$) oder 24 Transformer-Schichten ($\text{BERT}_{\text{LARGE}}$). Für jede Eingabe erzeugt das Modell am Ende jeder Schicht einen Output (Kodierung) pro Token, der eine semantische Kodierung darstellt. Diese Kodierung enthält Informationen über die Bedeutung und Kontext jedes Worts in Bezug auf den gesamten Text. Die erzeugte Kodierung besteht aus einem Vektor von numerischen Werten. Je nach Konfiguration des Modells besteht dieser Vektor zwischen 256 und 768 Einträgen. Diese Kodierungen können verwendet werden, um eine Vielzahl von NLP-Aufgaben auszuführen [DCLT18; TDP19], wie zum Beispiel Sentiment Analysis oder Named Entity Recognition.

Wir verwenden diese Kodierungen, um Text-Embeddings zu generieren. Wir nehmen in dieser Arbeit an, dass dann ähnliche Texte Vektoren (Text-Embeddings) erzeugen, welche ähnlich sind und daher im erzeugten Vektorraum nah beieinander liegen. Umgekehrt nehmen wir an, dass unähnliche Texte Vektoren erzeugen, die weit voneinander liegen.

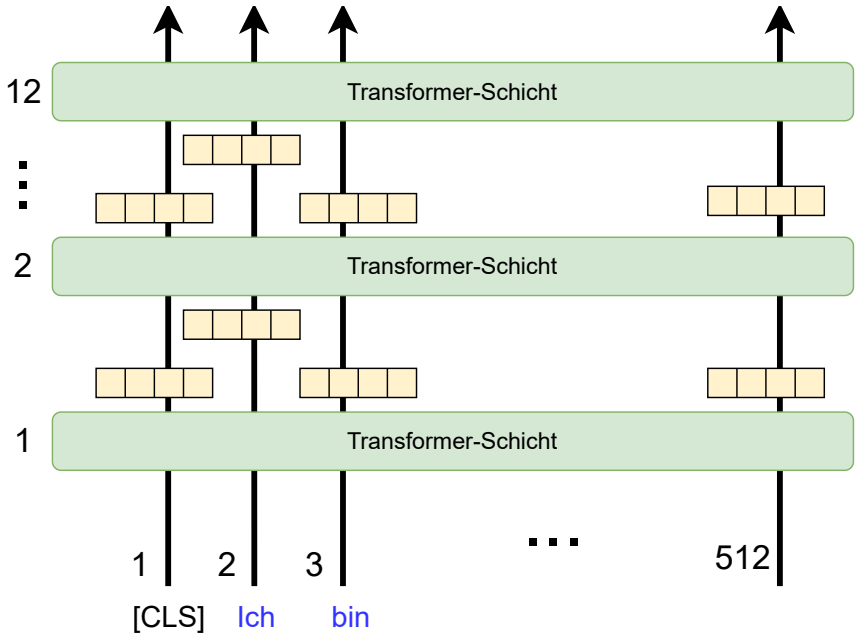

Abbildung 2.1: Die Transformer-Schichten von BERT und deren Output.

In Abbildung 2.1 ist die Verarbeitung von Text durch BERT visualisiert. Es ist zu erwähnen, dass bevor ein Text dem Modell als Input übergeben wird, der Text durch den *BERT-Tokenizer* tokenisiert wird. Standardmäßig wird dem Input ein **[CLS]** Token am Anfang des Inputs hinzugefügt, und ein **[SEP]** Token am Ende angefügt.

Das CLS-Token wird am Anfang jeder Eingabe verwendet und repräsentiert den Beginn des Dokuments. In [DCLT18] dient es als Symbol, das die Bedeutung des gesamten Dokuments repräsentiert.

Das SEP-Token wird zwischen zwei Sätzen oder zwei Abschnitten verwendet, um dem Modell anzuzeigen, dass sie getrennt sind. Dies hilft dem Modell, die Beziehungen zwischen den einzelnen Teilen einer Eingabe zu verstehen.

2.2 Active Learning

Active Learning ist eine Technik im Bereich des Machine Learnings, bei dem das System (oftmals das Modell) aktiv nach neuen Daten sucht, die es zur Verbesserung seiner Performance verwenden kann. Dabei werden dem System neue ihm unbekannte Daten vorgestellt, woraufhin das System Datenpunkte aussucht, die dann von einem Annotator zum Annotieren übergeben werden. Die ausgesuchten Datenpunkte sollen zur Verbesserung der Performance beitragen, und indirekt bewirken, dass weniger Datenpunkte verwendet werden müssen, um ein ähnlich leistungsfähiges Modell zu trainieren, als wenn man den kompletten Datensatz für Training herangezogen hätte [BU16].

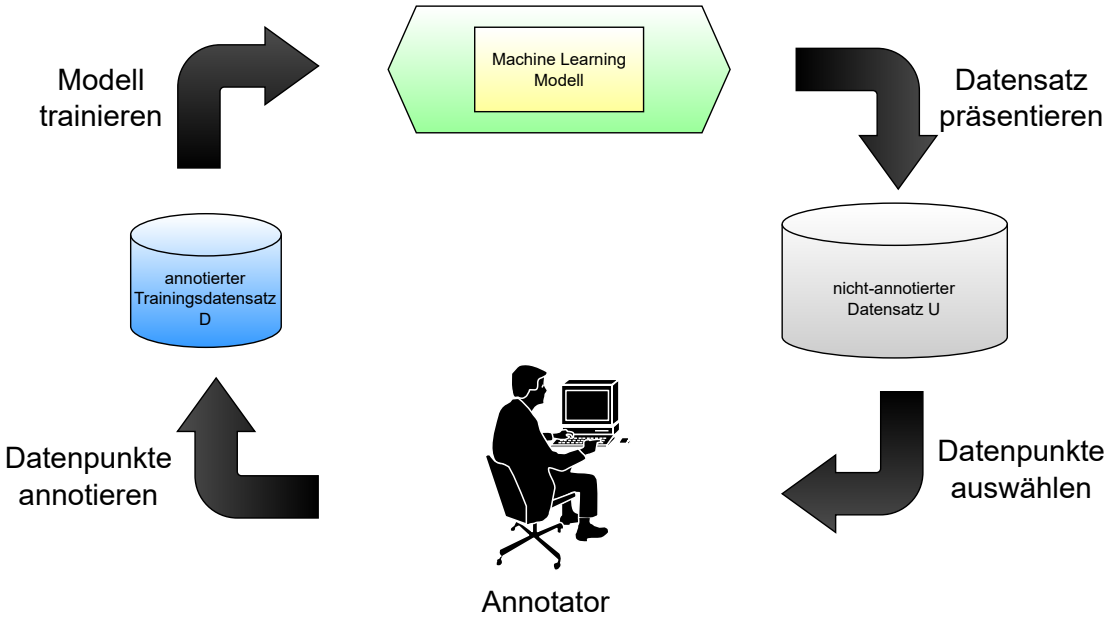

Abbildung 2.2: Der Active Learning Prozess nach [Set09].

2.2.1 Notwendigkeit von Active Learning

In den letzten Jahren hat das Gebiet der Künstlichen Intelligenz große Fortschritte gemacht. Gleichzeitig ist der Bedarf an großen Datenmengen enorm gestiegen, da der Zugang zu großen Datenmengen eine entscheidende Rolle bei der Entwicklung von KI-Modellen und der Verbesserung ihrer Leistung spielt. Denn je mehr Daten zum Trainieren des KI-Modells vorliegen, desto mehr kann es über die Natur seiner zu bearbeitenden Aufgabe lernen. Das führt zu einer Steigerung seiner Performance.

Im Bereich der natürlichen Sprachverarbeitung kommt es häufig vor, dass annotierte Daten nur in geringen Mengen vorliegen.

Ein Grund kann sein, dass zwar genügend Daten vorliegen, aber die Annotation der Daten sehr zeitaufwendig und in vielen Fällen somit auch kostenintensiv ist. Beispiele hierfür sind juristische und medizinische Texte [WBMK22]

In diesen Fällen kann Active Learning genutzt werden, um geeignete und informative Datenpunkte auszusuchen, diese von Menschen zu annotieren und dann mit ihnen Modelle trainieren. Daraus folgt, dass man ein ähnlich performantes KI-Modell erhält, aber durch die geringere Menge an nötigen Trainingsdaten weniger Zeit und Geld beim Annotieren dieser aufwendet.

2.2.2 Zufall vs. Strategie

Es stellt sich die Frage, warum man die zu annotierenden Datenpunkte nicht zufällig auswählt, da dies Zeit sparen würde, weil man von Anfang an eine größere Menge an Datenpunkte aussuchen und damit den Textklassifizierer trainieren könnte. Die Zeitersparnis kommt dadurch zustande, dass das Trainieren von Textklassifizierern mit kleineren Datenmengen wegfallen würde.

Ein potenzielles Problem ist, dass es dadurch wahrscheinlicher ist, dass Datenpunkte mit seltenen Klassen nicht ausgewählt werden und der dann zugrundeliegende Trainingsdatensatz seltene Klassen nicht genügend repräsentiert. Daraus resultiert eine verminderte Leistung von Machine Learning Algorithmen [HG09], da der Algorithmus nicht genügend Daten zu einer Klasse erlernt und somit für diese Klasse keine hohe Leistung erbringen kann.

Active Learning kann diesem Problem entgegenwirken, indem man Strategien und Methoden auswählt, die auf eine gleichmäßige Repräsentation der Datenklassen abzielt. Das hat den Vorteil, dass der ausgewählte Machine Learning Algorithmus gleich viele Datenpunkte von jeder Klasse zum Trainieren hat und somit keine Klasse bevorzugt wird.

2.2.3 Methoden des Active Learnings

Beim Active Learning gibt es eine große Menge an unterschiedlichen Methoden und Strategien, die man verfolgen kann [Set09].

Ein paar Beispiele sind:

- **Uncertainty Sampling:** Eine weitverbreitete Methode [LG94] bei der das Modell die Datenpunkte auswählt, bei denen es sich am wenigsten sicher ist welche Klasse sie haben. Diese Methode eignet sich am besten für probabilistische Modelle, weil ihre Ausgabe Aufschluss über die Sicherheit des Modells gibt.
- **Query-by-committee:** Ein Algorithmus [SOS92], der mehrere Modelle (Komitee) auf denselben annotierten Datensatz trainiert. Es werden dann die Datenpunkte ausgesucht, bei welchen das Komitee die wenigsten Übereinstimmungen bei der Klassifizierung hat.
- **Conformal Prediction:** Dieses Verfahren [GVV98; MSD12] sagt voraus, dass ein neuer Datenpunkt auf eine bestimmte Art und Weise eine ähnliche Klasse wie bekannte Datenpunkte haben wird. Der Grad der Ähnlichkeit zu den annotierten Daten wird verwendet, um das Vertrauen in die Vorhersage abzuschätzen. Dadurch ist es möglich, dass gezielt Datenpunkte mit bestimmten Klassen ausgesucht werden können, um beispielsweise ausgeglichene Datensätze zu erzeugen.

Mit den Entwicklungen der letzten Jahre im Bereich der Sprachmodelle ergeben sich neuartige Strategien und Methoden im Active Learning. In [TDP19] und [DCLT18] wird gezeigt, dass das Netzwerk des Sprachmodells *BERT* dazu verwendet werden kann, Aufgaben aus dem Bereich der maschinellen Sprachverarbeitung mit einer hohen Performance zu bearbeiten.

Damit wir untersuchen können, wie repräsentativ die von BERT erstellten Class-Embeddings sind, werden wir diese in einem Active Learning Setting untersuchen. Unsere Priorität ist, dass wir immer Datenpunkte auswählen, die zu einer Gleichverteilung der Klassen innerhalb des Trainingsdatensatzes führen. Damit wir dies erreichen, werden wir wie bei der *Conformal Prediction* die Ähnlichkeit zwischen unbekannten Daten und bekannten Daten messen. In unserem Szenario bedeutet das, dass wir die Ähnlichkeit zwischen den unbekannten Datenpunkten und den Class-Embeddings der bekannten Daten berechnen werden. Es werden dann die Datenpunkte ausgewählt, die am ähnlichsten zu den Klassen sind, welche am wenigsten im Trainingsdatensatz vertreten sind.

3 Methode

3.1 Intuition

Die Aufgabe unseres Modells ist die Textklassifikation auf einen Multi-Class Multi-Label Datensatz. Nach dem Training ist das Modell mit den zugrundeliegenden Daten bekannt, und ist bereit, die Zugehörigkeit eines (unbekannten) Textes T zu einer Reihe vordefinierter Klassen C zu bestimmen. Wenn man das Modell als *Black Box* betrachtet, dann kann man sagen, dass es den tokenisierten Text T als Eingabe bekommt, und die Klassen $\{c \mid c \in C\}$ als Ausgabe ausgibt. Unterschiedliche Eingaben ergeben unterschiedliche Ausgaben. Man kann also annehmen, dass nur der Text T die entscheidenden Informationen beinhaltet, die die Ausgabe beeinflussen und somit die Klassenzugehörigkeit von T bestimmen. Somit sollten die Klassen von C im Text erkennbar sein.

Eine Möglichkeit, Text in einem kontinuierlichen Vektorraum darzustellen, besteht in *Embeddings*. Das sind Vektoren, die dazu konditioniert sind, Textstücken entsprechen zu können.

BERT ist ein neuronales Netzwerk mit mehreren inneren Schichten, die jeweils unterschiedliche Outputs generieren, wenn das Modell eine Eingabe verarbeitet. Wenn also nur der Text T die entscheidenden Informationen beinhaltet, die die Ausgabe beeinflussen, dann müssen die während der Berechnung entstandenen Outputs charakteristisch für die Eingabe sein. Also kann man diese Outputs als Embedding T_e für einen eingegebenen Text T verwenden, um die Texte in den Vektorraum zu überführen. Da diese Outputs dafür verwendet werden, um die Klassen eines Textes T zu berechnen, kann man davon ausgehen, dass die Ouputs mit den jeweiligen Klassen zusammenhängen. Daraus folgt, dass T_e auch mit den Klassen zusammenhängt.

Da T_e sowohl von der Eingabe T als auch von dessen Klassen abhängt, ist die Annahme, dass T_es, die derselben Klassen angehören, Ähnlichkeiten in ihrem Text aufweisen. Ähnlichkeiten können sich im Vektorraum unter anderem mit der Entfernung von Vektoren zeigen. Somit nehmen wir an, dass T_es, die derselben Klassen angehören, im Vektorraum auch näher beieinander liegen. Daher kann man davon ausgehen, dass wenn ein neuer Text T^* in denselben Vektorraum abgebildet wird, er wahrscheinlicher dieselben Klassen wie seine Nachbarn besitzt.

An dieser Stelle definieren wir das Zentroid einer Menge von T_e derselben Klasse c als das Class-Embedding der Klasse c. Das Class-Embedding der Klasse c ist der Mittelpunkt

aller T_e, die zur Klasse c gehören. Das Class-Embedding fungiert als Repräsentant für die Klasse c.

Wir verwenden die Class-Embeddings, um eine Klasse von T^* vorherzusagen, indem wir schauen zu welchem Class-Embedding T^* am nächsten ist. Wir nehmen an, dass die ausgewählte Klasse von T^* derselben Klasse entspricht, wie die von dem nächsten Class-Embedding.

3.2 Erstellung der Embeddings

Für die Erstellung der Embeddings der Texte extrahieren wir die jeweils entstehenden Vektoren von den einzelnen Transformer-Schichten des BERT-Modells. Wir verwenden hierfür die entstehenden Vektoren von dem CLS-Token, da dieser auch in [DCLT18] für NLP-Tasks als Repräsentant des gesamten Dokuments verwendet wird. In Abbildung 3.1 wird die Extraktion der entstehenden Vektoren abgebildet, wenn eine Eingabe getätigt wird. Man sieht, dass nur die Outputvektoren der Transformerschichten extrahiert werden, deren Ursprung sich im CLS-Token befindet.

Nachdem wir die Vektoren erhalten, ergeben sich viele Möglichkeiten diese miteinander zu kombinieren, um ein Document-Embedding zu erhalten. Für die Erstellung der Embeddings setzen wir auf dieselben Strategien wie im Abschnitt "*Feature-based Approach with BERT*" in [DCLT18].

Diese sind:

- **Sum 12 Layers**: Alle 12 Vektoren werden miteinander addiert.
- **Sum Last 4 Layers**: Die letzten vier Vektoren (9, 10, 11,12) werden miteinander addiert.
- **Concat Last 4 Layers**: Die letzten vier Vektoren (9, 10, 11, 12) werden der Länge nach konkateniert.
- **Last Layer**: Der letzte entstehende Vektor (12) wird verwendet.
- **2nd to Last Layer**: Der vorletzte Vektor (11) wird verwendet.

Die **Sum 12 Layers** Strategie ist die einzige Strategie, welche alle Outputvektoren für die Berechnung der Embeddings verwendet.

Die Strategien **Sum Last 4 Layers** und **Concat Last 4 Layers** verwenden beide die letzten vier Outputvektoren. Sie unterscheiden sich lediglich in der Kombination dieser. Von allen hier genannten Strategien erzielte **Concat Last 4 Layers** in [DCLT18] die beste Performance im angeführten "Named Entity Recognition" Task.

Die beiden letzten Strategien **Last Layer** und **2nd to Last Layer** setzen lediglich auf ein Outputvektor.

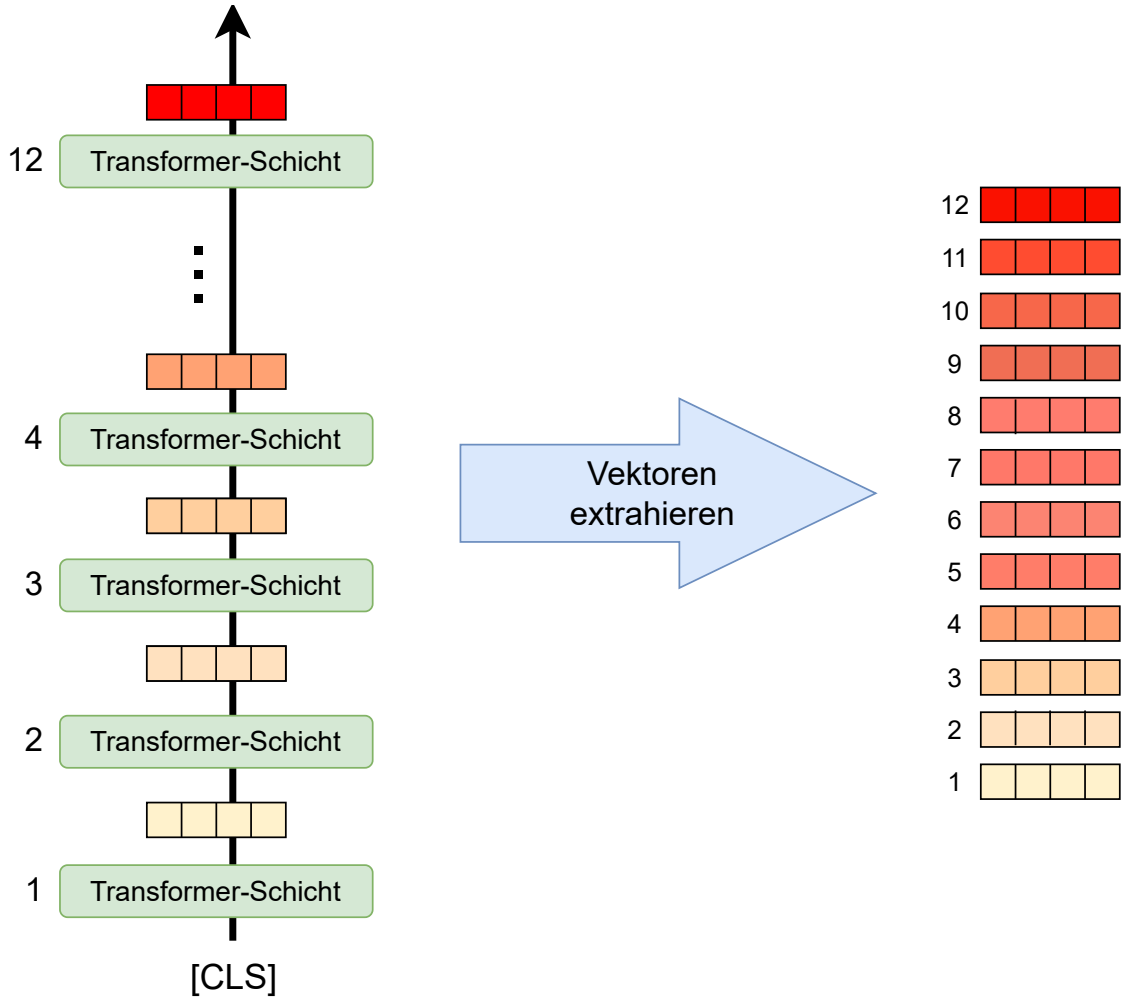

Abbildung 3.1: Extraktion der Vektoren.

3.3 Active Learning mit Class-Embeddings und BERT

Unser Active Learning Prozess ist ein zyklischer, überwachter Lernmechanismus, der strategische Datenpunkte auswählt, die von einem menschlichen Annotator beschriftet werden sollen und dann dem Modell zum Trainieren gegeben werden.

Unsere Strategie zielt darauf ab, eine gleichmäßige Verteilung der Klassen im Trainingsdatensatz D zu garantieren, um den Textklassifizierer mit seltenen Klassen vertraut zu machen.

Der Ablauf des Active Learnings kann dann in drei wiederholenden Schritten zusammengefasst werden:

1. Trainieren des Klassifikationsmodells M auf verfügbare Daten D.
2. Modell wählt Datenpunkten aus U aus, deren angenommene Klasse am wenigsten in D vertreten ist
3. Übergabe an den Annotator, Annotation der Datenpunkte und Hinzufügen zu D.

Wir wissen die Klassen der ungelabelten Dokumente vorher nicht. Um diese abschätzen zu können, gehen wir davon aus, dass Dokumente derselben Klasse Embeddings besitzen, die im aufgespannten Vektorraum nah beieinander sind. Um die Nähe eines Textes zu einer Klasse berechnen zu können, verwenden wir Class-Embeddings, die stellvertretend für die jeweilige Klasse stehen. Für die Nähe eines Embeddings zu einer Klasse wird dann die euklidische Norm als Abstandsmaß verwendet.

Ein Class-Embedding C_e zu einer bestimmten Klasse c wird berechnet, indem man das Zentroid aller T_e berechnet, deren T zu c gehört.

$$C_e = \{mean(T_e) | T \in D \text{ und } T \text{ gehört zu } c\}$$

Im Active Learning-Setting berechnen wir C_e anhand des aktuellen Datensatzes D und wählen dann k Datenpunkte aus, die den C_es am nächsten sind, deren Klasse im Trainingsdatensatz am wenigsten vertreten ist. Wir aktualisieren und evaluieren M nach der Auswahl von k Datenpunkten und wiederholen diesen Prozess, bis das Annotationbudget aufgebraucht ist.

Diese Auswahl von Datenpunkten wird an einem Beispiel in Abbildung 3.2 dargestellt.

Der vollständige Ablauf ist im Algorithmus 3.1 detailliert beschrieben.

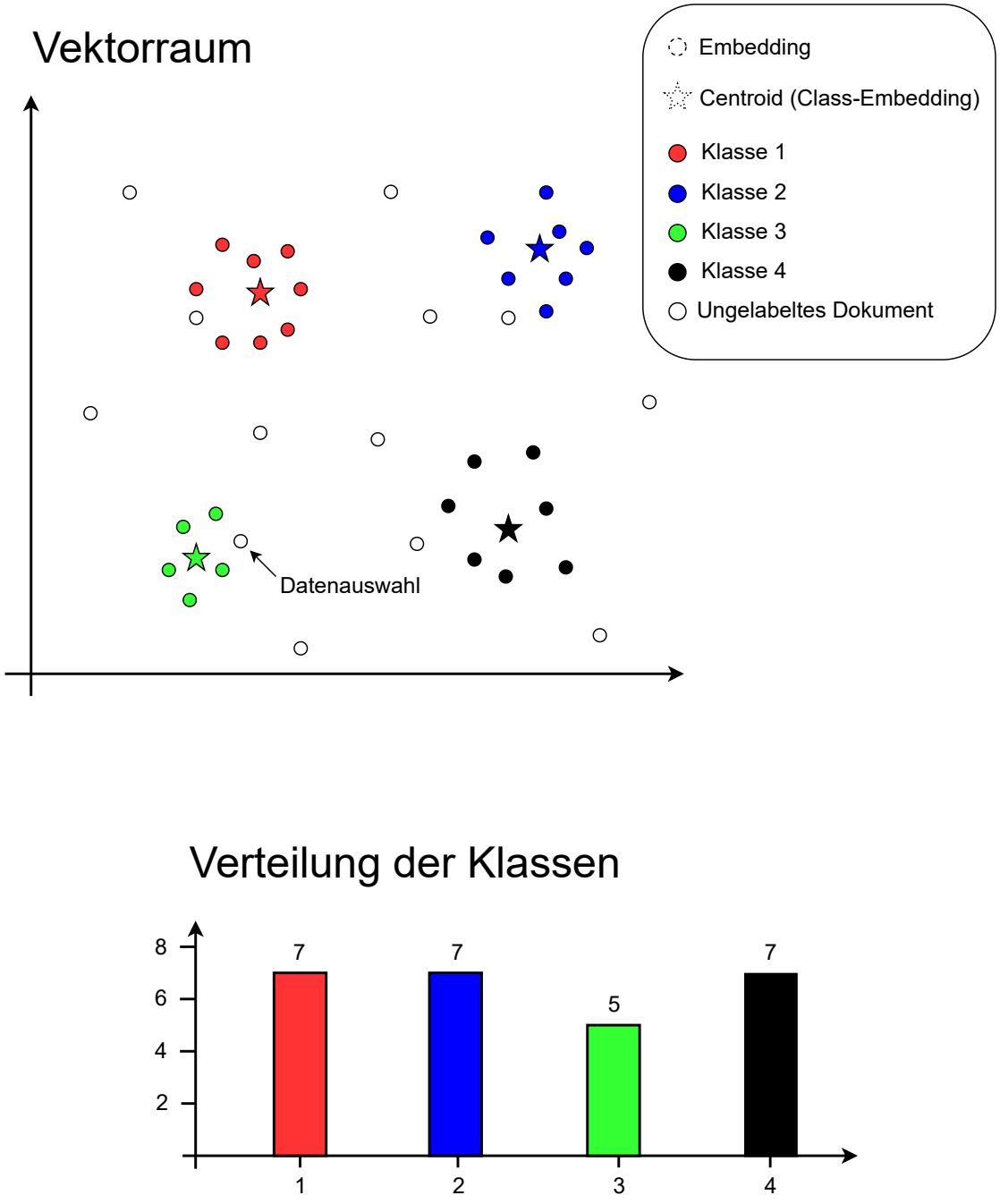

Abbildung 3.2: In der Datenauswahlphase wird das Dokument ausgewählt und einer Klasse zugeordnet, das am nächsten zu dem Class-Embedding ist, dessen Klasse am wenigsten im Datensatz vertreten ist. Die angenommene Klasse des Dokuments entspricht dann der Klasse des Class-Embeddings.

Algorithmus 3.1 Active Learning mit Class Embeddings und BERT

procedure AL-CE-BERT(labeled set D, unlabeled set U, budget b, sample size k)
  while budget > 0 **do**
    model M $\leftarrow$ new BERTModel()
    train M on D
    $C_e \leftarrow$ Compute Class Embeddings on D from M
    $k^* \leftarrow k$
    while $k^* > 0$ **do**
      $c_{min} \leftarrow$ least frequent class in D
      $T \leftarrow T \epsilon U, T_e$ closest to C_e of c_{min}
      annotate T
      $D \leftarrow D \cup T$
      $k^* \leftarrow k^* - 1$
      $b \leftarrow b - 1$
    end while
  end while
end procedure

4 Experiment

4.1 Evaluation

Wir werden die vorgestellten Strategien zum Erstellen von Embeddings (siehe Abschnitt 3.2 und Abschnitt 4.2) evaluieren. Da jede Strategie zu unterschiedlichen Embeddings und Class-Embeddings führt, kann dies die Zusammensetzung des Trainingsdatensatzes D beeinflussen. Dies ist so, weil sich mit unterschiedlichen Embeddingstrategien die Auswahl neuer Datenpunkte ändern kann.

Um die Auswirkung der unterschiedlichen Strategien zu messen, wird die Evaluation der Ergebnisse in den folgenden vier Teile unterteilt.

4.1.1 Macro F1-Score

Zuerst untersuchen wir die Performance des Textklassifizierers abhängig von der Größe seines Trainingsdatensatzes. Für die Messung der Performance wird der erreichte Macro F1-Score des Textklassifizierers auf den Testdatensatz gemessen. Der Macro F1-Score wird benötigt, da der zugrundeliegende Datensatz ein Multi-Class Multi-Label Datensatz ist, und nicht alle Klassen gleichhäufig vorkommen. Beim Macro F1-Score werden für alle Klassen der F1-Score berechnet und dann der Durchschnitt berechnet. Das hat zur Folge, dass alle Klassen gleich gewichtet werden, und somit seltene Klassen nicht benachteiligt werden.

Die Formel für den Macro F1-Score lautet:

$$\text{Macro F1-Score} = \frac{2 \cdot \text{Macro Precision} \cdot \text{Macro Recall}}{\text{Macro Precision} + \text{Macro Recall}}$$

, wobei Macro Precision und Macro Recall wie folgt berechnet werden:

$$\text{Macro Precision} = \frac{\text{Precision}_1 + \text{Precision}_2 + ... + \text{Precision}_n}{n}$$

$$\text{Macro Recall} = \frac{\text{Recall}_1 + \text{Recall}_2 + ... + \text{Recall}_n}{n}$$

n ist hierbei die Anzahl der Klassen.

Der Micro F1-Score ist für unseren Fall nicht optimal, da für seine Berechnung auf die Gesamtzahl von *true positives*, *false postives* und *false negatives* gesetzt wird. Das hat zur Folge, dass seltene Klassen weniger Einfluss auf den Score haben als häufig vorkommende Klassen.

Die Formel für den Micro F1-Score lautet:

$$\text{Micro F1-Score} = \frac{2 \cdot \text{Micro Precision} \cdot \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}}$$

,wobei Micro Precision und Micro Recall wie folgt berechnet werden:

$$\text{Micro Precision} = \frac{\sum \text{true positives}}{\sum \text{true positives} + \sum \text{false positives}}$$

$$\text{Micro Recall} = \frac{\sum \text{true positives}}{\sum \text{true positives} + \sum \text{false negatives}}$$

4.1.2 Treffer

Da jede Runde des Active Learning Prozesses neue Datenpunkte ausgewählt werden, die annotiert werden sollen, messen wir, wie oft die angenommene Klasse eines Datenpunktes auch die tatsächliche Klasse ist. Wird ein Datenpunkt ausgesucht, der die vermeintliche Klasse enthält, wird dies als ”Treffer” bezeichnet. Auf diese Art und Weise können wir überprüfen, ob unsere Intuition der Realität entspricht. Da jede Runde 50 Datenpunkte ausgesucht werden, ist das Maximum, das erzielt werden kann 50 Treffer.

4.1.3 Entropie der Verteilung

Wir werden die Verteilung der Klassen in den Trainingsdatensätzen untersuchen. Das Ziel unserer Active Learning Prozesses ist Trainingsdatensätze zu erzeugen, in denen die Vorkommnisse aller Klasse ungefähr gleichverteilt sind. Um messen zu können, wie ebenmäßig alle Klassen verteilt sind, verwenden wir das Entropiemaß. In der Informationstheorie ist die Entropie [Sha48] ein Maß für die Unbestimmtheit einer Zufallsvariablen. Wenn wir die Vorkommnisse der einzelnen Klassen als Zufallsvariable ansehen, können wir die Entropie der Verteilung berechnen. Je näher dann der berechnete Wert der Entropie dem theoretisch maximalen Wert ist, desto gleichverteilter sind die Vorkommnisse der Klassen.

Die Entropie wird wie folgt berechnet:

$$\text{Entropie } H = -\sum_i p(c_i) \log_2 p(c_i)$$

, wobei $p(c_i)$ die Wahrscheinlichkeit für das Vorkommen der Klasse c_i in der Verteilung ist.

Diese berechnet sich so:

$$p(c_i) = \frac{n_{c_i}}{\sum_{c_j \in C} n_{c_j}}$$

,wobei n_{c_i} die Anzahl der Vorkommnisse der Klasse c_i ist.

In unserem Beispiel verwenden wir einen Datensatz mit 10 Klassen. Der maximale Wert, der sich dann für die Entropie ergibt ist:

$$H_{\text{MAX}} = -\sum_i 0.1 \log_2 0.1 = -10 \cdot 0.1 \log_2 0.1 = -\log_2 0.1 \approx 3.3219$$

4.1.4 Abstand der Class-Embeddings zueinander

Da der Kern unserer Active Learning Strategie die errechneten Class-Embeddings sind, möchten wir auch die Ähnlichkeit unterhalb derer messen. Ähnlich wie bei der Datenselektion messen wir die Ähnlichkeit der Class-Embeddings zueinander anhand ihrer Abstände mit der euklidischen Norm im Vektorraum.

Der euklidische Abstand zweier n-dimensionaler Vektoren p und q ist wie folgt definiert:

$$d(p,q) = ||q-p||_2 = \sqrt{(q_1-p_1)^2 + ... + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

Die Annahme ist, dass Klassen, die semantisch ähnlich sind, auch ähnliche Class-Embeddings besitzen, und somit im Vektorraum nah beieinander sind. Semantisch unterschiedliche Klassen sollten dann im Umkehrschluss zu größeren Abständen der Class-Embeddings führen. Somit können wir untersuchen, ob der Abstand von den Class-Embeddings zueinander Rückschlüsse auf einen semantischen Unterschied der jeweiligen Klassen erlaubt.

4.2 Baseline - Vergleich mit Zufall und Pretrained BERT

Um die Interpretierbarkeit der Ergebnisse zu steigern werden wir das gleiche Experiment durchführen, aber anstelle der Datenselektion durch Class-Embeddings, werden wir zufällig 50 Datenpunkte pro Active Learning Runde auswählen (**Random Selection**).

Des Weiteren untersuchen wir, wie sich das pretrained BERT-Modell ohne zusätzliches Fine-Tuning und lediglich unter Verwendung der bereits vorhandenen pretrained Gewichtungen bei dieser Aufgabe verhält. Hierfür werden wir die **Sum 12 Layers** Strategie anwenden, um Embeddings mit dem pretrained BERT-Modell zu berechnen (**Pretrained-BERT**)

4.3 Verwendeter Datensatz

Der für diese Arbeit verwenden Datensatz ist derselbe, der in [WBMK22] verwendet wurde. Dabei handelt es sich um eine modifizierte Version des Eurlex57K-Korpuses (bezeichnet als eurlex), das Auszüge des europäischen Rechts enthält. Der ursprüngliche Korpus ist mit mehreren hundert Klassen annotiert und ist für die multi-label Texterkennung gedacht. Das bedeutet, dass ein Text (Datenpunkt) zu einer beliebigen Anzahl von Klassen gehören kann. Für diese Arbeit haben wir eine reduzierte Version verwendet, welche 5 häufige Klassen und 5 seltene Klassen enthält. Dadurch bleibt die Multi-Label-Natur des Datensatzes erhalten.

Die enthaltenen Klassen sind:

import, *export refund*, *pip fruit*, *fruit vegetable*, *citrus fruit*,
quantitative restriction, *Germany*, *Portugal*, *ship's flag*, *export licence*

Ein Textklassifizierer, der mit allen Daten trainiert wurde, erzielte einen Macro F1-Score von 0.94.

4.4 Setup

Wir verwenden $\text{BERT}_{\text{BASE}}$ (12 Transformer-Schichten und 768 Einträge pro Outputvektor von den Transformer-Schichten) für die Textklassifikation mit einem *Dropout Layer* und einem *Linear Layer*. Wir trainieren das Modell für 15 Epochen mit *Early Stopping*, einer Batchgröße von 11 und einer adaptiven Lernrate mit zusätzlicher Gewichtsdämpfung (*ADAMW*).

Für die Dokumenten-Embeddings verwenden wir die Outputvektoren der Transformerschichten (siehe Abschnitt 3.2) von der BERT-Instanz von dem zuvor trainierten Textklassifizierer.

Wir simulieren den Active Learning Prozess, indem wir einen Teil des Korpus als gelabelten Datensatz D und den Rest als ungelabelten Datensatz U ansehen. Ein Datenpunkt gilt somit als gelabelt, wenn er sich im Datensatz D befindet. Wenn also ein Datenpunkt aus dem Datensatz U von dem Modell ausgewählt wird, wird dieser durch das Hinzufügen zu D annotiert.

Wir starten mit einem initialen Datensatz D von 100 zufällig ausgewählten Datenpunkten, die eine Gleichverteilung der vorkommenden Klassen ergeben. Wir fragen in jedem Active Learning Schritt 50 Datenpunkte ab, bis das Annotationen-Budget von 700 Datenpunkten ausgeschöpft ist.

Alle Experimente wurden auf einer NVIDIA GeForce RTX 2080 Ti GPU durchgeführt.

5 Ergebnisse

Im Folgenden werden die Ergebnisse von allen Experimenten vorgestellt. Dabei stellt jedes Experiment jeweils eine Methode bei der Erstellung des Embeddings dar.

Zu jedem Experiment präsentieren wir den **Macro F1-Score** des Textklassifizierers abhängig von Trainingsdatengröße, weil uns unter anderem interessiert, wie sich die Datenauswahl auf die Performance eines Textklassifizierer auswirkt.

Unser Hauptinteresse gilt aber der Performance der Class-Embeddings, und daher werden sich die restlichen Visualisierungen auf die gewählten Datenpunkte und die damit erzeugten Trainingsdatensätzen konzentrieren.

Es wird die Anzahl der richtig ausgewählten Datenpunkte pro Runde (**Treffer**) im Active Learning Prozess gezeigt, und wie die Anzahl der Treffer von der Größe des Trainingsdatensatzes und von der gewählten Embeddingmethode abhängt.

Außerdem wird die **Entropie der Verteilung** der Klassen im Trainingsdatensatz abhängig von seiner Größe gezeigt.

Um einen Eindruck zu erhalten, wie sich die Entropie der Verteilung ergibt, wird die Verteilung der Klassen in den Trainingsdatensätzen gezeigt, die von der **"Random Selection"**-Methode, der **"Pretrained BERT"**-Methode und der **"Sum 12 Layers"**-Methode erzeugt wird.

Um die tatsächliche semantische Ähnlichkeit der Klassen mit den Abständen von den Class-Embeddings zu vergleichen, werden wir die Abstände der Class-Embeddings zueinander mit einer **Abstandsmatrix** visualisieren. Dafür verwenden wir die Class-Embeddings, die bei der **"Sum 12 Layers"**-Methode entstanden sind.

Alle Visualisierungen wurden mit dem Python-Modul *matplotlib* erstellt.

5.1 Macro F1-Score

In Abbildung 5.1 sind die erzielten Macro F1-Scores von den Textklassifizierern, in Abhängigkeit von der Größe ihrer Datensätze, dargestellt. In dieser Abbildung werden die Performances von den Textklassifizierern von Wertz et al. und dieser Arbeit dargestellt. Für beide Arbeiten sind jeweils der zufalls- und (SENTENCE-)BERT-basierte Ansatz abgebildet

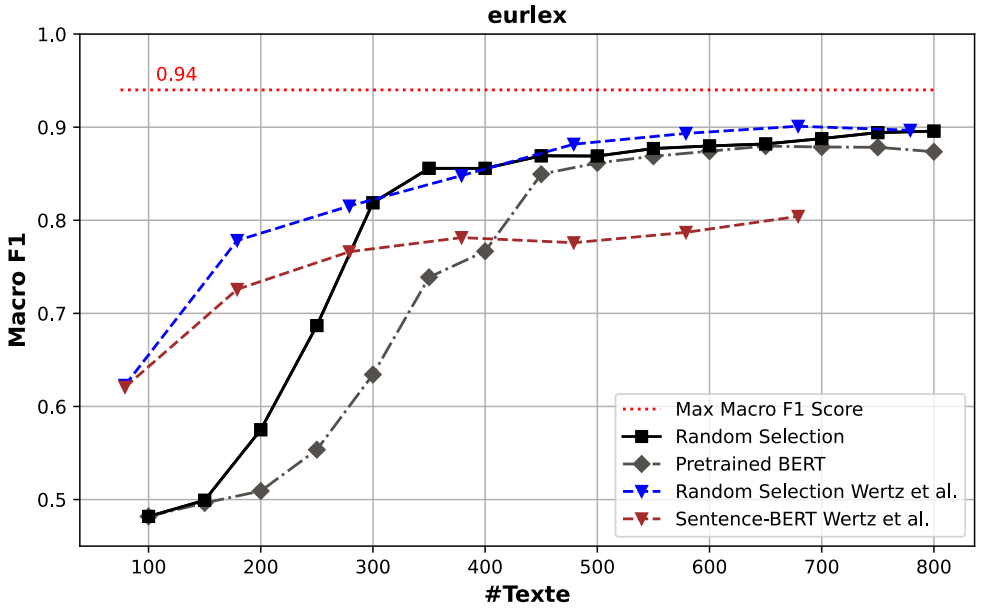

Abbildung 5.1: Vergleich der Macro F1-Scores der trainierten Textklassifizierer abhängig von der Größe des Trainingsdatensatzes und der Datenselektionsmethode während des Active Learning Prozesses. Hier werden unsere zufallsbasierten und BERT-basierten Ansätze mit Wertz et al. verglichen.

Man sieht, dass die Performance bei Wertz et al. zu Beginn bei 0.62 liegt und somit höher ist als die Performance von dieser Arbeit, welche bei 0.48 liegt.

Bei einer Datensatzgröße von 300 sieht man, dass beide zufallsbasierten Ansätze ähnliche Ergebnisse liefern. Auch im weiteren Verlauf sieht man, dass beide Ansätze ähnliche Ergebnisse liefern und gegen den theoretischen Maximalwert von 0.94 konvergieren.

Man sieht, dass unser BERT-basierte Ansatz zu Beginn langsamer anwächst als der zufallsbasierte Ansatz, aber den Unterschied bis 450 Datenpunkten im Datensatz aufholt und dann mit einem Macro F1-Score von 0.85 eine ähnliche Performance erzielt. Ab dann ist Performance äquivalent zu dem zufallsbasierten Ansatz und konvergiert auch gegen den in der Praxis erzielten Maximalwert.

Lediglich der auf SENTENCE-BERT-basierte Ansatz von Wertz et al. scheint bei einem Macro F1-Score von 0.78 bis 0.8 zu stagnieren. Es muss jedoch erwähnt werden, dass die Daten bei dem SENTENCE-BERT Ansatz nur bis zu einer Datensatzgröße von 679 vorliegen.

In Abbildung 5.2 sind die erzielten Macro F1-Scores von den Textklassifizierern abgebildet, deren Datensätze durch die von uns gewählten Methoden erstellt werden.

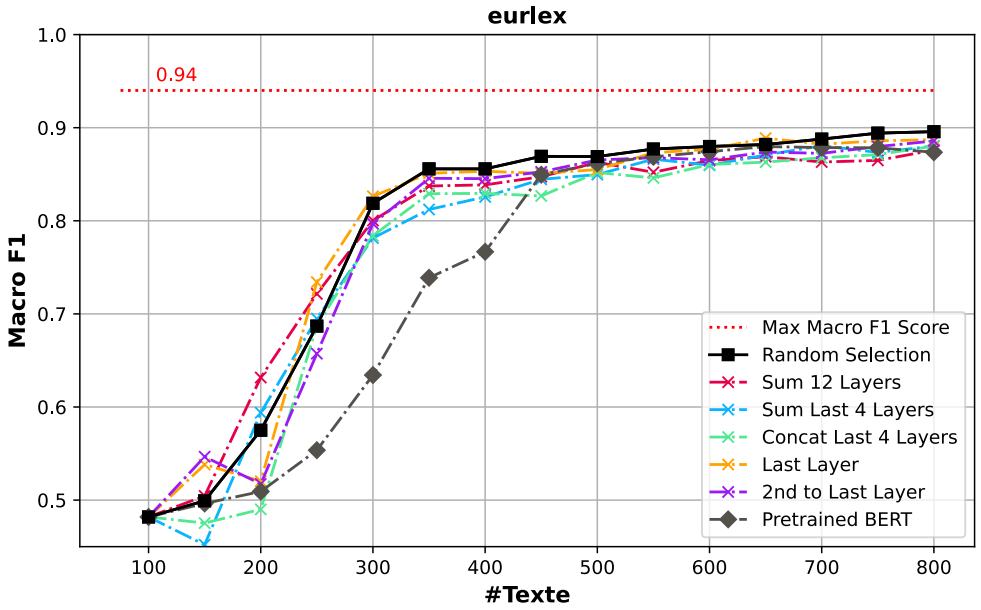

Abbildung 5.2: Vergleich der Macro F1-Scores der trainierten Textklassifizierer in Bezug auf die Trainingsdatensatzgröße und die Datenselektionsmethode während des Active-Learning-Prozesses, die auf unseren verschiedenen Strategien zur Erstellung der Embeddings basieren.

Man sieht, dass bei allen Methoden, mit Ausnahme von der Pretrained-BERT Methode, der Verlauf des Macro F1-Scores bis auf kleine Abweichungen ähnlich ist. Alle Methoden beginnen mit demselben Macro F1-Score von 0.48. Bis zu einer Datensatzgröße von 300 Texten wächst der Macro F1-Score am stärksten bis zu einem Wert von 0.81. Ab da steigt die Performance schwächer als zuvor und nähert sich mit wachsendem Trainingsdatensatz dem maximalen Macro F1-Score von 0.94. In unserer Arbeit endet das Active Learning bei einer Größe von 800 Datenpunkten und bei dieser Größe des Datensatzes wird der beste Macro F1-Score von 0.9 erreicht.

Man sieht, dass die Performance bei dem Pretrained-BERT Ansatz langsamer anwächst als bei den anderen Ansätzen. Da erreicht man erst bei 450 Datenpunkten einen ähnlichen Macro F1-Score (0.85) wie die anderen Ansätzen. Ab dann ist der Verlauf ähnlich wie bei den anderen Ansätzen.

5.2 Treffer

Abbildung 5.3 veranschaulicht die Anzahl der Treffer in Abhängigkeit von der Datensatzgröße, auf der das Modell trainiert wird. Als Treffer werden die Übereinstimmungen der angenommenen und tatsächlichen Klassenzugehörigkeit von Datenpunkten im Datenselektionsschritt bezeichnet. Da in jeder Runde 50 Datenpunkte ausgewählt werden, ist somit die maximale Anzahl an Treffer pro Runde 50.

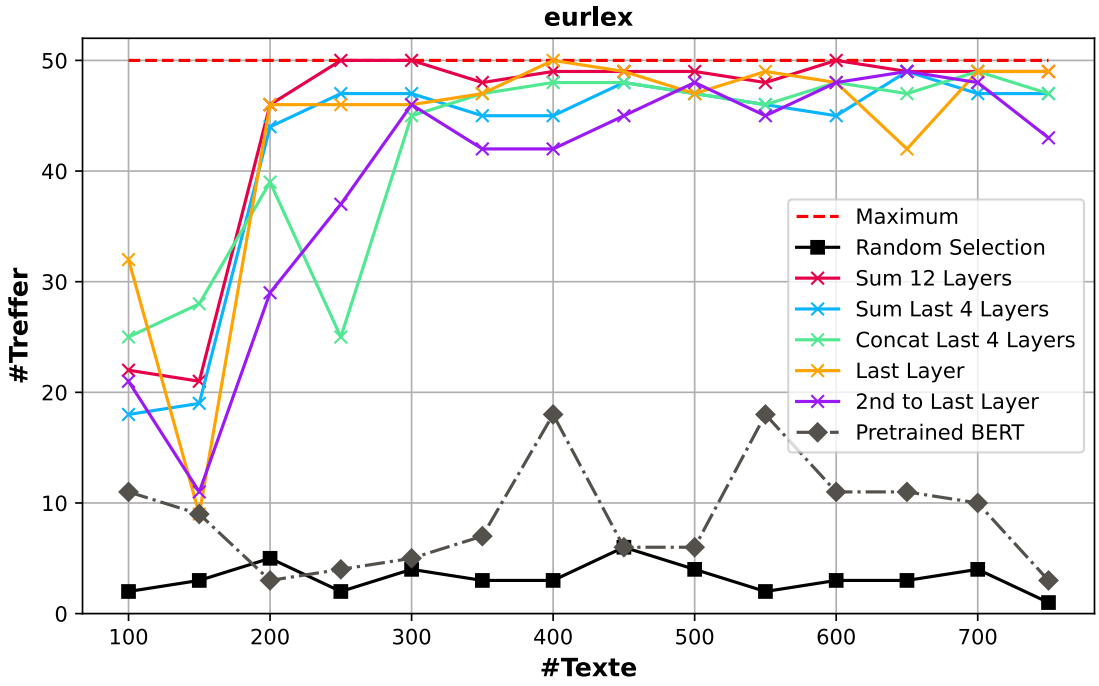

Abbildung 5.3: Anzahl der Treffer im Active Learning Prozess.

Für die zufallsbasierte Methode sei erwähnt, dass hier keine Klassen angenommen werden und somit keine Treffer laut gegebener Definition erzielt werden. Für diese Methode werden die ausgewählten Datenpunkte, die zum Zeitpunkt des Datenselektionsschrittes eine Klasse haben, die am geringsten im Datensatz vorkommt, als Treffer angesehen.

Man sieht, dass bei der zufälligen Auswahl von Datenpunkten die Anzahl der Treffer im Vergleich zu den anderen Methoden am geringsten ist, und im gesamten Verlauf einen Wert zwischen 1 und 6 annehmen.

Die Pretrained-BERT Methode erzielt im Vergleich zu den anderen Strategien auch wenige Treffer. Bis auf zwei Ausreißer (18 Treffer) befinden sich die Trefferanzahl bei dieser Methode zwischen 3 und 11 Treffer. Damit performt es ähnlich wie die zufallsbasierte Methode, und weist somit eine niedrigere Performance als die anderen Methoden.

Die Methoden, die darauf setzen, dass BERT mit den Daten trainiert wird und dann Class-Embeddings aus den Transformerschichten generiert werden, weisen eine höhere Performance auf als der Zufall und pretrained BERT. Die Trefferanzahl beginnt beim initialen Datensatz mit Werten zwischen 18 und 32. Bei allen Methoden steigen die Trefferanzahl bis zu einem Trainingsdatensatz mit 300 Datenpunkten. Ab einer Trainingsdatengröße von 300 erzielen alle Methoden mit ihren Class-Embeddings 42 oder mehr Treffer.

Die beste Performance erbringt die ”Sum 12 Layers” Methode, die bereits mit einem Datensatz von 250 oder mehr Datenpunkten stets 48 oder mehr Treffer erzielt.

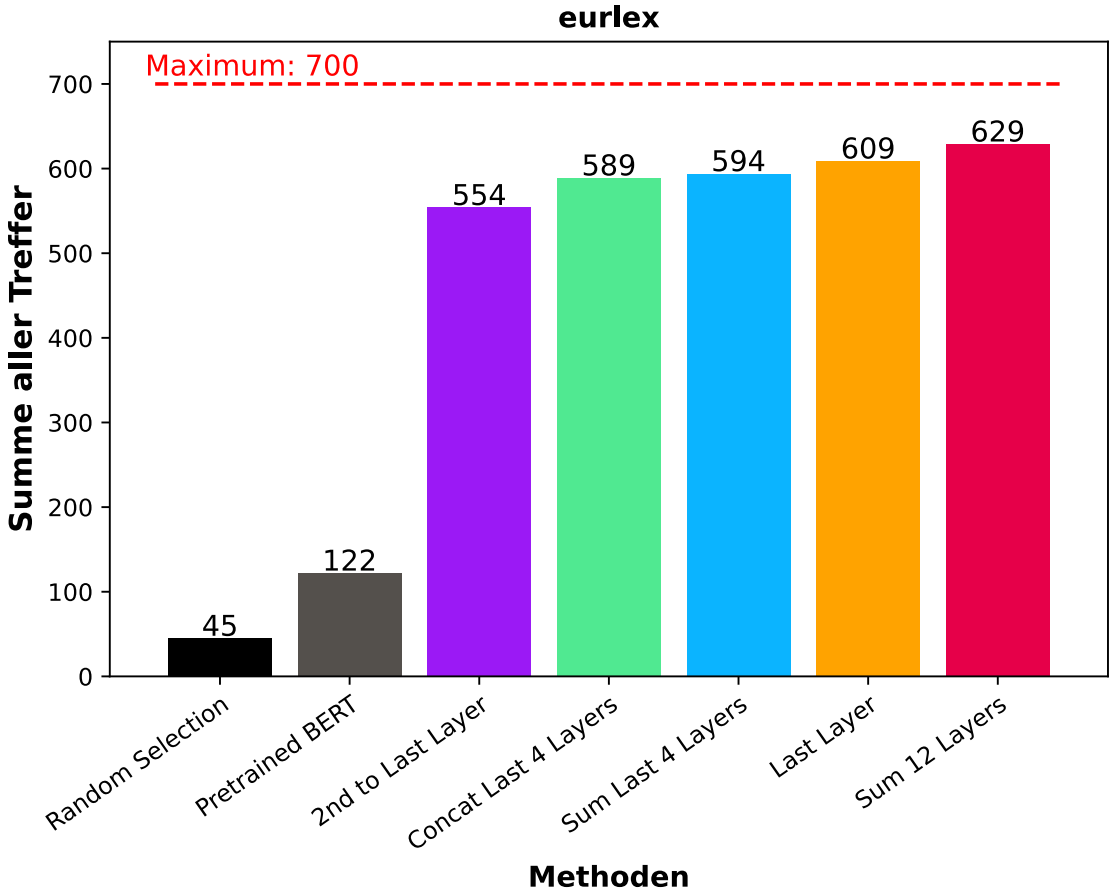

Abbildung 5.4: Summe aller Treffer während des Active Learning Prozesses.

In Abbildung 5.4 sieht man die summierten Treffer aller Methoden während des Active Learning Prozess. Hier wird die hohe Performance der ”Sum 12 Layers” Methode verdeutlicht, denn sie erzielt im gesamten Experiment 629 Treffer von maximal 700 möglichen Treffer. Außerdem sieht man, dass die Methoden, die auf ein fine-tuned BERT-Modell setzen, deutlich mehr Treffer erzielen, als pretrained BERT und die zufällige Datenselektion.

5.3 Entropie der Klassenverteilung

In Abbildung 5.5 wird der Verlauf der Entropie der Klassenverteilung im Trainingsdatensatz in Abhängigkeit von dessen Größe gezeigt. Wie bereits in Abschnitt 4.1.3 erwähnt, ist das theoretische Maximum der Entropie für 10 Klassen bei circa 3.3219.

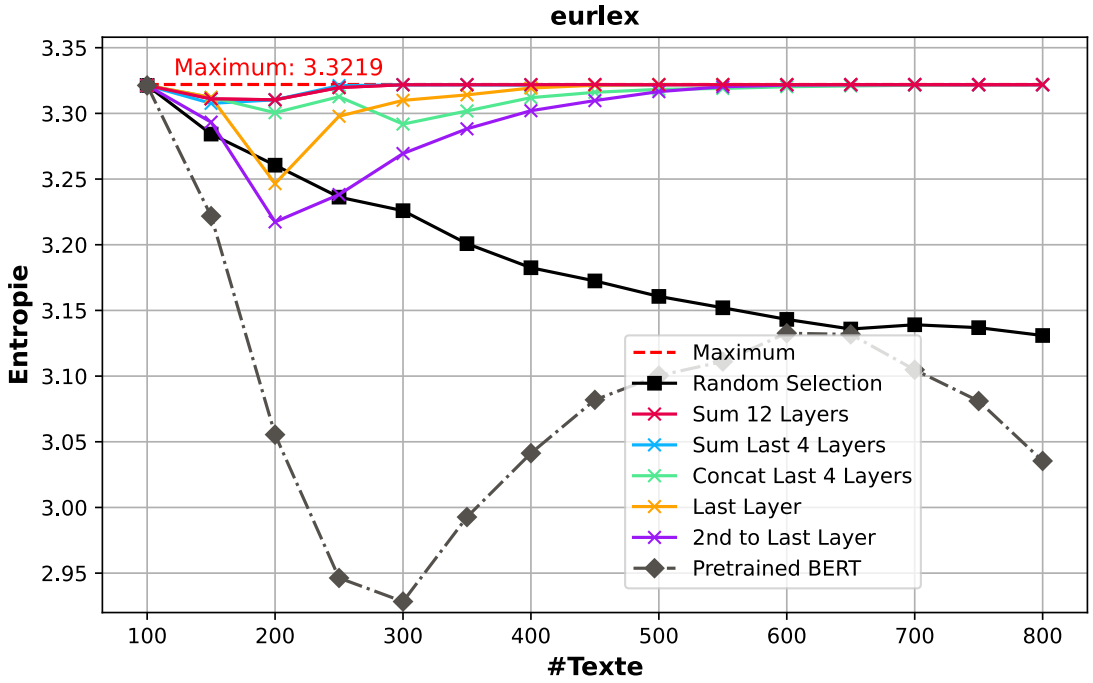

Abbildung 5.5: Die Entropie der Verteilung der Klassen abhängig von ihrer Größe und deren Datenselektionsmethode.

Man sieht, dass bei der zufälligen Datenselektion die Entropie mit wachsendem Datensatz stetig sinkt.

Bei der Pretrained BERT Methode sinkt die Entropie von allen Methoden am stärksten. Sie steigt wieder zwischen 300 und 650, bevor sie wieder sinkt. Im gesamten Verlauf ist der Wert Entropie für alle Methoden am niedrigsten, was in den ungleichverteiltesten Datensatz resultiert.

Zu Beginn sinkt jedoch der Wert von der Entropie bis zu einer Datensatzgröße von 200. Der Wert der Entropie bei der ”Last Layer” und ”2nd to Last Layer” Methode sinkt dabei stärker als die Entropie bei der zufälligen Datenselektion. Für alle Methoden gilt, dass ab 200 die Trefferanzahl im Datenselektionsschritt steigt und das bewirkt, dass die Entropie steigt und sich dem theoretisch maximalen Wert nähert.

5.4 Verteilung der Klassen

Damit man ein besseres Bild über die resultierenden Verteilungen der Klassen in den Datensätzen bekommt, sind in Abbildung 5.6 bis Abbildung 5.9 die Verteilungen der Klassen, die durch die Methoden ”Random Selection”, ”Pretrained BERT” und ”Sum 12 Layers” entstehen, dargestellt. Da für alle Experimente derselbe initiale Datensatz verwendet wird, welcher eine fast ausgeglichene Klassenverteilung aufweist, beginnen alle Methoden mit derselben Klassenverteilung.

Bei der ”Sum 12 Layers” Methode sieht man die hohe Performance der Class-Embeddings. Unabhängig von der Datensatzgröße sind alle Verteilungen bis auf kleine Abweichungen sehr ausgeglichen.

Im Gegensatz dazu sieht man, dass die ”Pretrained BERT” Methode zwei Klassen bevorzugt und fast ausschließlich Datenpunkte aus diesen auswählt. Die beiden Klassen, die bevorzugt werden, sind ”export refund” (3568) und ”ship’s flag” (2282). Klasse (3568) ist eine Klasse, die im Gesamtdatensatz häufig vorkommt. Klasse (2282), die am häufigsten im resultierenden Trainingsdatensatz vertreten ist, ist eine der seltenen Klassen im gesamten Datensatz.

Bei der zufallsbasierten Methode sieht man, dass mit wachsender Größe des Trainingsdatensatzes, der Aufbau des Gesamtdatensatzes sichtbar wird. Schon ab 350 Datenpunkten (davon 250 zufällig ausgewählt) sieht man, dass der Gesamtdatensatz aus 5 häufigen Klassen und 5 seltenen Klassen besteht. Je mehr Datenpunkte zu dem Trainingsdatensatz hinzugefügt wird, desto mehr wird der Unterschied von den häufigen und seltenen Klassen deutlich.

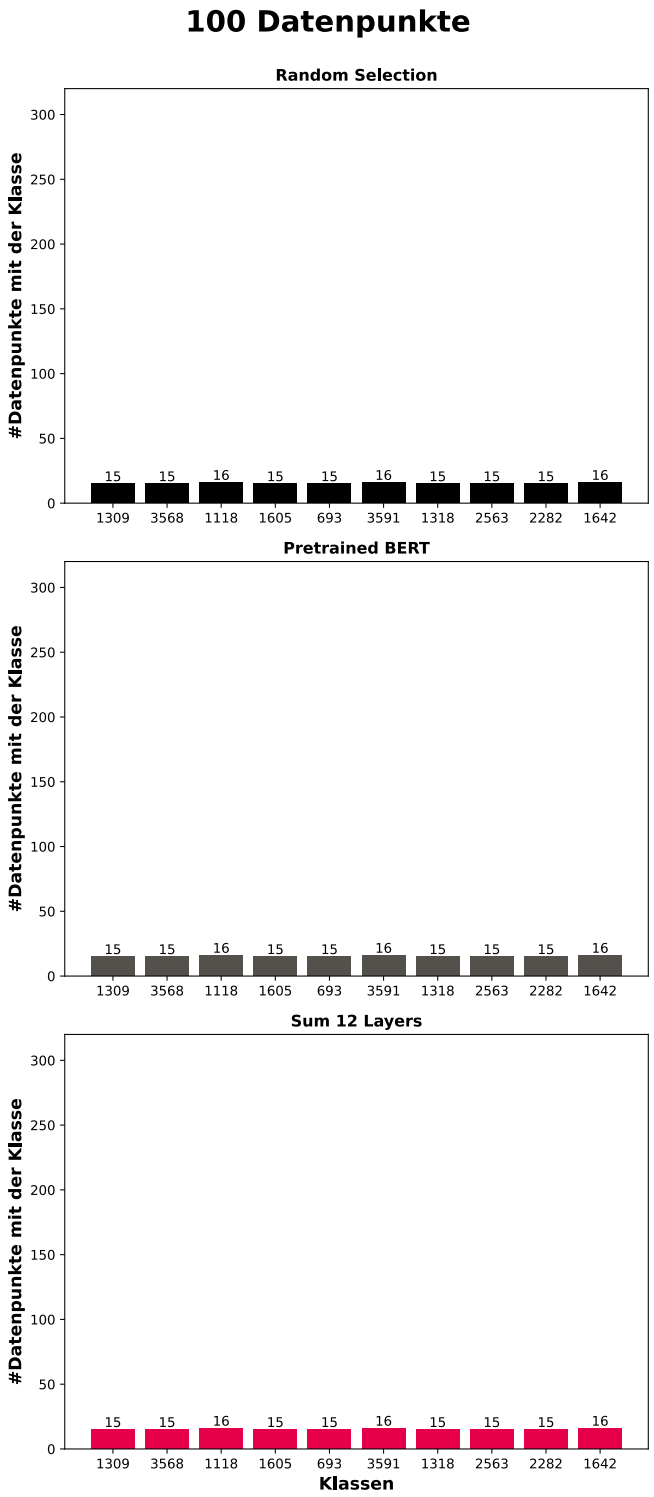

Abbildung 5.6: Verteilung der Klassen im Trainingsdatensatz mit 100 Datenpunkten.

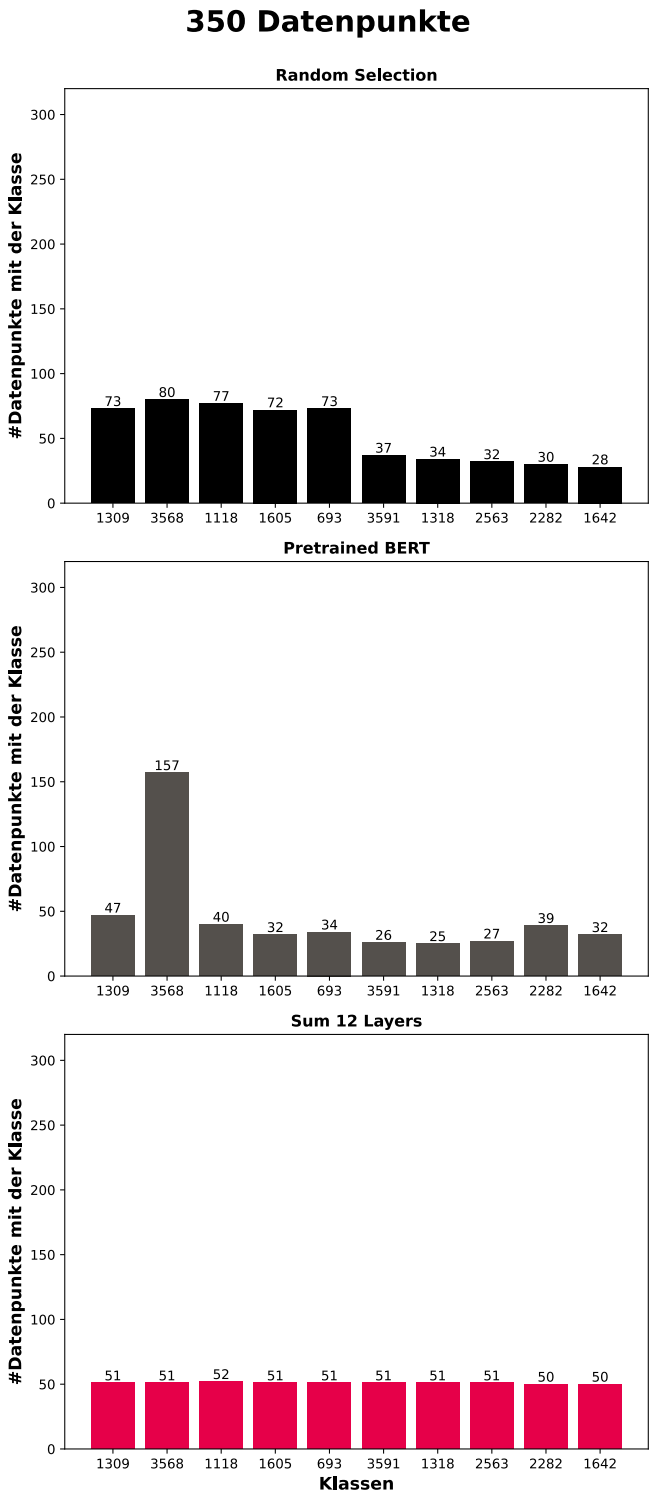

Abbildung 5.7: Verteilung der Klassen im Trainingsdatensatz mit 350 Datenpunkten.

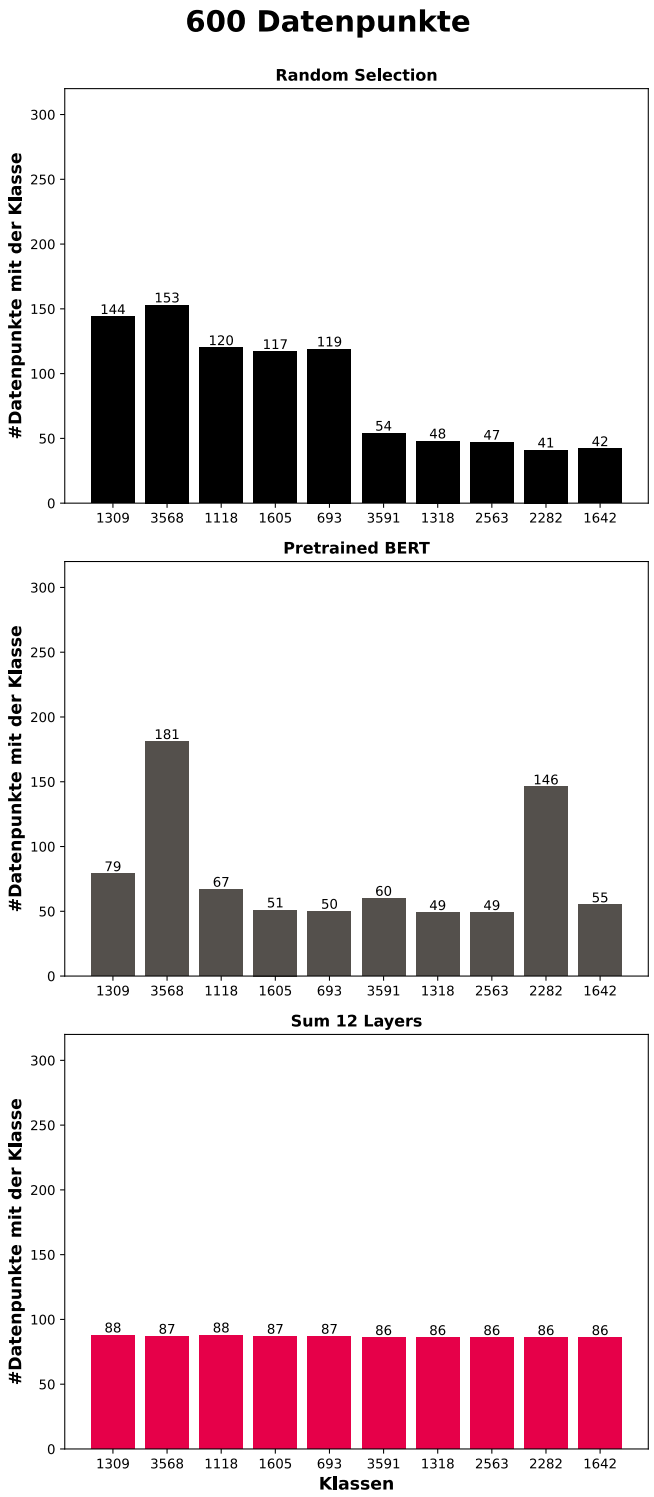

Abbildung 5.8: Verteilung der Klassen im Trainingsdatensatz mit 600 Datenpunkten.

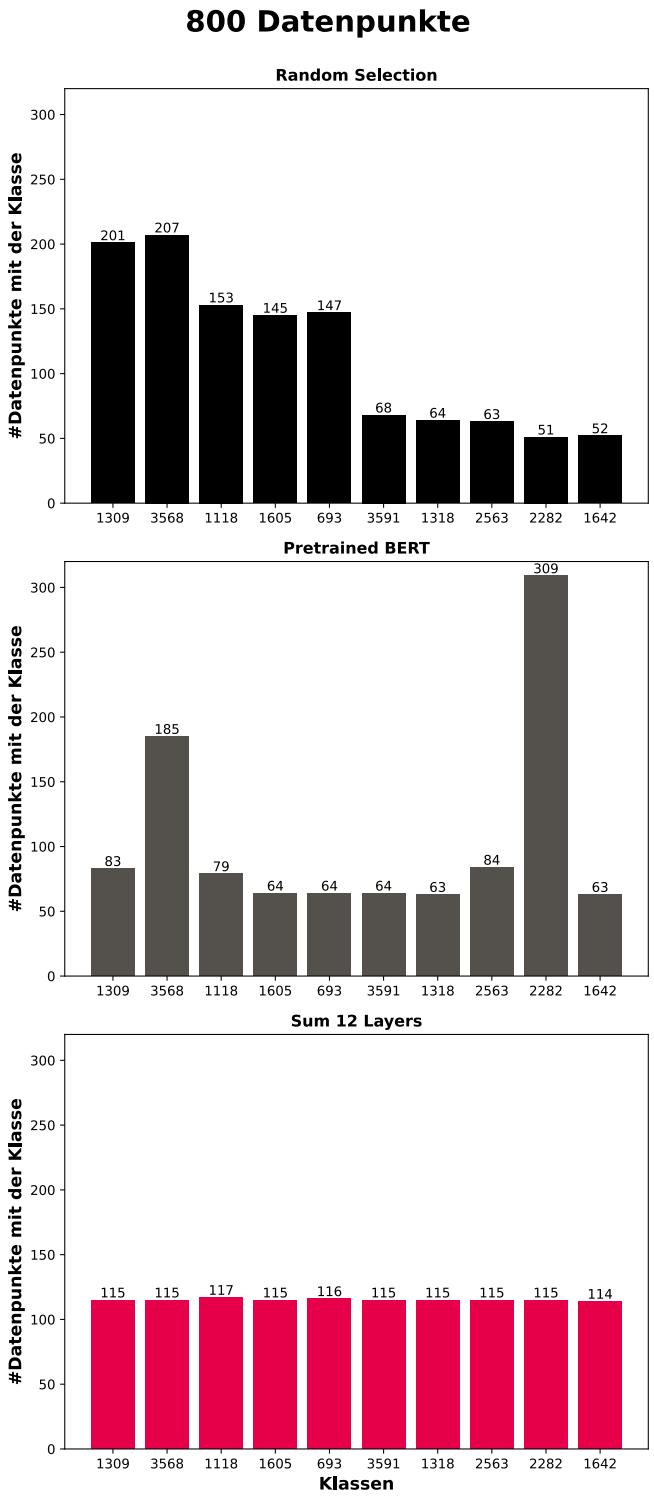

Abbildung 5.9: Verteilung der Klassen im Trainingsdatensatz mit 800 Datenpunkten.

5.5 Abstandsmatrix

In Abbildung 5.10 sind die resultierenden Abstände von den Class-Embeddings, die bei der ”Sum 12 Layers” Methode entstehen. Die Zeilen und Spaltenindizes sind die Klassen, und jeder Eintrag gibt den Abstand von den jeweiligen Indizes (Klassen) zueinander an. Für den Abstand wird die euklidische Norm verwendet und das Ergebnis nach dem Komma abgeschnitten, sodass nur ganze Zahlen resultieren. Die verwendeten Farben repräsentieren hierbei den Wert für den Abstand. Je kleiner ein Wert ist, desto blauer ist die verwendete Farbe, und je größer Wert ist, desto roter ist die verwendete Farbe für die Zelle.

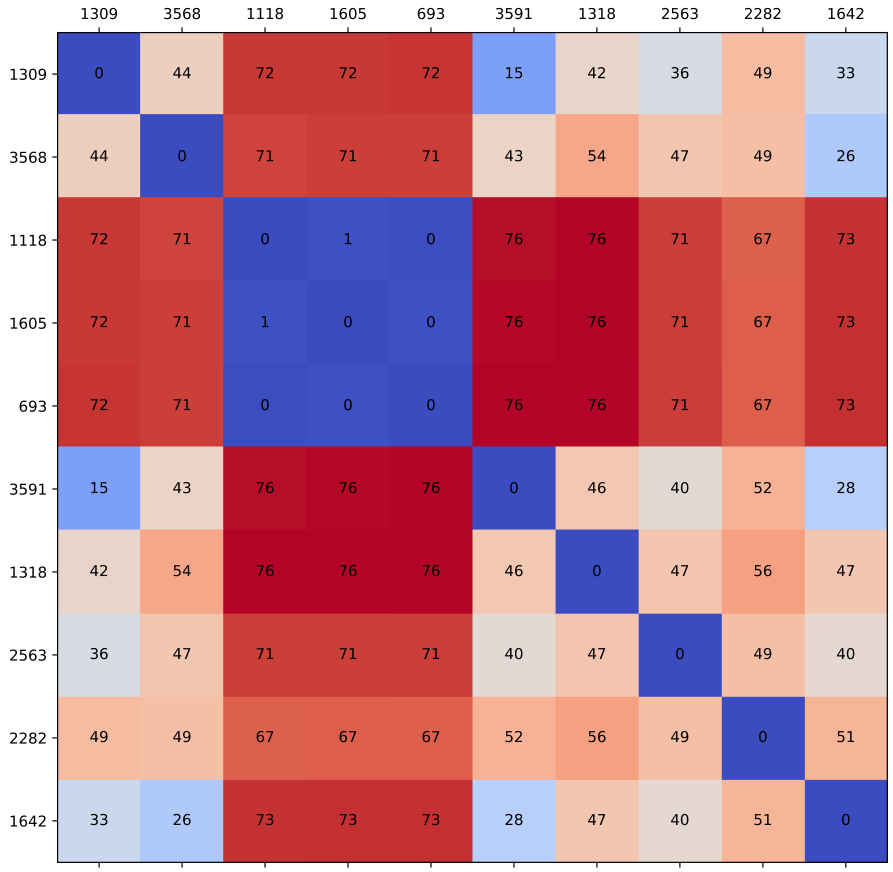

Abbildung 5.10: Abstände der in der ”Sum 12 Layers” Methode resultierenden Class-Embeddings zueinander.

Um den semantischen Sinn der jeweiligen Klassen zu verstehen, sind in Tabelle 5.1 die jeweiligen Klassenbedeutungen angeführt.

Klassezahl	Bedeutung
1309	import
3568	export refund
1118	pip fruit
1605	fruit vegetable
693	citrus fruit
3591	quantitative restrictions
1318	Germany
2563	Portugal
2282	ship's flag
1642	export licence

Tabelle 5.1: Die Klassenzahlen und deren Bedeutung.

6 Diskussion

In diesem Kapitel werden die erzielten Ergebnisse aus Kapitel 5 interpretiert.

6.1 Macro F1-Score

6.1.1 Vergleich zu Wertz et al.

In Abbildung 5.1 sind die erzielten Macro F1-Scores von den Textklassifizierern, in Abhängigkeit von der Größe ihrer Datensätze, dargestellt. In dieser Abbildung beabsichtigen wir unsere Ergebnisse mit den Ergebnissen von Wertz et al. zu vergleichen. Es werden jeweils die zufallsbasierten Ansätze miteinander verglichen und die Ergebnisse, die auf Embeddings von untrainierten BERT-Modellen setzen. Es sei erwähnt, dass Wertz et al. das SentenceBERT Sprachmodell verwendet, während wir das Basissprachmodell von BERT verwenden.

Die unterschiedlichen Startwerte für die Macro F1-Scores sind wahrscheinlich auf unterschiedliche Hyperparameter der Textklassifizierer und auf unterschiedlichen initialien Datensätzen zurückzuführen. Denn wenn man die zufallsbasierten Ansätzen betrachtet, sieht man, dass sie bereits bei einer Datensatzgröße von 300 ähnliche Ergebnisse liefern. Im weiteren Verlauf liefern beide Ansätze ähnliche Ergebnisse und konvergieren gegen den in der Praxis erzielten Maximalwert von 0.94. Daraus kann man schließen, dass wenn genügend Daten vorhanden sind, beide Klassifizierer eine ähnliche Performance aufweisen.

Unser BERT-basierter Ansatz liefert zu Beginn, bei einer kleinen Größe des Trainingsdatensatzes, einen schwach performenden Textklassifizierer. Werden jedoch genügend Daten ausgewählt und dem Trainingsdatensatz hinzugefügt, erzielt der Textklassifizierer ähnliche Ergebnisse wie die der zufallsbasierten Ansätze. Die anfängliche mindere Performance lässt sich mit einem unausgewogenen Datensatz begründen. In Abbildung 5.7 wird gezeigt, dass zu Beginn die meisten Datenpunkte einer Klasse zugehörig sind. Dadurch wird der Klassifizierer nicht genügend mit anderen Daten trainiert, und das führt zu einer schwächeren Performance. In Abbildung 5.8 und Abbildung 5.9 sieht man, dass zwar zwei Klassen bei der Datenselektion bevorzugt werden, aber auch Datenpunkte von den anderen Klassen hinzugefügt werden, und dies für den Klassifizierer genügend ist, um in der Lage zu sein, Datenpunkte der anderen Klassen richtig zu klassifizieren.

Der SentenceBERT-Ansatz von Wertz et al. liefert im Vergleich dazu einen Textklassifizierer, welcher einen Macro F1-Score von circa 0.8 erzielt. Dieses Ergebnis ist bedingt vergleichbar mit den anderen F1-Scores-Kurven, da die Ergebnisse bei diesem Ansatz bei einer Datensatzgröße von 679 aufhören. Dennoch lässt sich bis zu dieser Datensatzgröße sagen, dass dieser Ansatz einen schwächer performenden Klassifizierer erzeugt, als die bisher genannten Ansätze. In [WBMK22] wird dies mit der unzureichenden Datenauswahl begründet, da diese einen unausgewogenen Trainingsdatensatz erzeugt.

Abgesehen von dem SentenceBERT-Ansatz lässt sich aus diesen Ergebnissen ableiten, dass für die Performance des Textklassifizierers bei ausreichender Datenmenge (ab 450) kein signifikanter Unterschied ergibt, welche Methode man für die Datenselektion wählt. Daraus folgt, dass unser Klassifizierer nicht inhärent schwächer ist als der von Wertz et al. verwendete Klassifizierer.

Abbildung 5.1 verifiziert zudem, dass unsere Methode in dieser Arbeit der Methode von [WBMK22] folgt, da die beiden zufallsbasierten Ansätze ab einer Datengröße von 300 Datenpunkten eine ähnliche Performance aufweisen. Somit kann man unseren zufallsbasierten Ansatz als Baseline für weitere Experimente verwenden.

6.1.2 Unsere Strategien

Abbildung 5.2 zeigt die erreichten Macro F1-Scores der Textklassifizierer, die mit Datensätzen trainiert werden, welche mithilfe der von uns generierten Embeddings erstellt werden.

Abgesehen von der Pretrained-BERT Methode, lassen die Ergebnisse in Abbildung 5.2 darauf schließen, dass die Wahl der Methode für die Datenselektion keinen signifikanten Unterschied beim Macro F1-Score macht. Die ähnliche Performance der Textklassfizierers bei unseren Strategien, ist auf eine ähnlich hohe Performance der Embeddings zurückzuführen. Trotz unterschiedlicher Strategien zur Generierung der Embeddings, zeigt Abbildung 5.5, dass nahezu durchgehend ein ausgeglichener Datensatz erzeugt wird. Wir nehmen aufgrund der ähnlichen Klassenverteilung in den Trainingsdatensätzen an, dass die Textklassifizierer deswegen äquivalente Ergebnisse erzielen. Ein Beleg für diese Annahme ist der Verlauf des Macro F1-Scores des Pretrained-BERT Ansatzes. In Abbildung 5.6 bis Abbildung 5.9 sieht man, dass die Klassenverteilung des Datensatzes sehr unausgeglichen ist, was zu einem langsameren Anstieg des Macro F1-Scores des Textklassifizierers zur Folge hat.

An dieser Stelle kann man sich fragen, wieso der zufallsbasierte Ansatz auch so performante Leistungen erzielt wie unsere Strategien, da bei diesem Ansatz ähnlich wie bei dem Pretrained-BERT Ansatz ein unausgeglichener Datensatz resultiert. Die ähnliche Leistung des Textklassifizierers kann daher kommen, dass die zufällige Auswahl von Datenpunkten einen Datensatz erstellt, welcher die relative Verteilung der Klassen aus

dem gesamten Datensatz nachahmt. Somit erlernt der Klassifizierer die zugrundeliegende Verteilung der Klassen und labelt die Daten der Wahrscheinlichkeitsverteilung des gesamten Datensatzes nach.

Dennoch ist zu beachten, dass bei anderen Experimenten der zufallsbasierte Ansatz auf Datensätzen mit einer großen Anzahl von Klassen möglicherweise nicht so effektiv ist. Dies liegt daran, dass es dann länger dauert, bis alle Klassen ausreichend repräsentiert sind.

Insgesamt zeigt sich, dass unsere BERT-Methoden in Bezug auf den Macro F1-Score keine signifikante Verbesserung gegenüber einer zufälligen Datenauswahl erzielen. Dies, in Kombination mit dem erhöhten Rechenaufwand beim Erstellen und Trainieren von Zwischenklassifizierern, legt nahe, dass unsere Active Learning Strategie im Hinblick auf den Macro F1-Score nicht von Vorteil ist. Unsere Active Learning Strategie dient in erster Linie jedoch zur systematischen Auswertung der erstellten Class Embeddings. Dadurch ist es uns möglich, verschiedene Ansätze zur Erstellung von Embeddings zu untersuchen. Wir können nicht nur die Leistung der Embeddings in Abhängigkeit von der Menge der Trainingsdaten analysieren, sondern auch ihre Performance bei der Verarbeitung unbekannter Daten untersuchen. Es wird deutlich, dass die untersuchten BERT-Methoden in Bezug auf den F1-Score weitgehend gleich effektiv sind, was darauf schließen lässt, dass die erstellten Embeddings eine ähnliche Performance bei der Datenselektion haben. Dies wird im Folgenden untersucht.

6.2 Treffer

Abbildung 5.3 veranschaulicht die Anzahl der Treffer in Abhängigkeit von der Datensatzgröße, auf der das Modell trainiert wird. Als Treffer wird die Übereinstimmung der angenommenen und tatsächlichen Klassenzugehörigkeit von Datenpunkten im Datenselektionsschritt bezeichnet.

Bei der zufälligen Auswahl von Datenpunkten ist die Anzahl der Treffer im Vergleich zu den anderen Methoden am geringsten. Das Ergebnis ergibt Sinn, da wir einen Datensatz vorliegen haben, welcher aus 10 Klassen besteht. Fünf dieser Klassen kommen häufig vor und die anderen fünf Klassen seltener. Wenn nun der Trainingsdatensatz im Active Learning Prozess unausgeglichen wird, und deshalb ein Datenpunkt mit einer seltenen Klasse verlangt wird, um den Datensatz auszugleichen, ist die Wahrscheinlichkeit klein, dass wir zufällig so einen Datenpunkt auswählen. So lässt sich auch die geringe Trefferanzahl in jeder Runde erklären.

Die Pretrained-BERT Methode erzielt im Vergleich zu den anderen Strategien auch wenige Treffer. Eine mögliche Ursache dafür ist, dass der vorliegende Datensatz europäische Gesetzestexte enthält, für die die pretrained Version von BERT möglicherweise nicht ausreichend trainiert ist.

Die Methoden, die darauf setzen, dass BERT mit den Daten trainiert wird und dann Class-Embeddings aus den Transformerschichten generiert werden, erzielen die meisten Treffer. Ab einer Datensatzgröße von 300 werden 42 oder mehr Treffer erzielt, was nahe am Maximum von 50 ist.

Die Ergebnisse lassen darauf schließen, dass die erstellten Class-Embeddings repräsentativ für ihre jeweilige Klasse sind, und dazu verwendet werden können, die Klassenzugehörigkeit von ungelabelten Texten mithilfe des Abstandes zu ihren Embeddings abzuleiten. Hierbei wichtig zu erwähnen ist, dass diese performanten Ergebnisse erst ab einer gewissen Menge an Trainingsdaten erzielt werden. Obwohl die Trefferanzahl zu Beginn jedoch geringer ist, übertrifft sie dennoch die des Zufalls und der Pretrained-BERT Methode. Daraus lässt sich schließen, dass die gewählte Methode effektiv ist und mit weiteren Daten optimale Ergebnisse erzielt. Mit zunehmender Datenmenge erhalten die Systeme mehr Informationen, was schließlich zu den besten Class-Embeddings führt.

Die beste Performance erbringt die ”Sum 12 Layers” Methode, die bereits ab einer Datensatzgröße von 250 stets 48 oder mehr Treffer erzielt. Eine mögliche Erklärung hierfür ist, dass dieser Ansatz alle 12 Outputvektoren der Transformerschichten von BERT miteinander addiert und somit vereint. Die dann entstehenden Embeddings beinhalten die maximal mögliche Menge an Informationen aus den Transformerschichten. Dadurch ist es möglich, dass die Information aus allen zwölf Schichten hier hilft, repräsentative Embeddings zu erzeugen.

In Abbildung 5.4 sieht man die summierten Treffer aller Methoden während des Active Learning Prozess. Hier wird die bereits erwähnte hohe Performance der ”Sum 12 Layers” Methode verdeutlicht. Außerdem sieht man, dass die Methoden, die auf ein fine-tuned BERT-Modell setzen, deutlich mehr Treffer erzielen, als Pretrained-BERT und die zufällige Datenselektion.

Diese Ergebnisse lassen andeuten, dass ein Lernprozess stattfindet, welcher in repräsentative Class-Embeddings führt. Die Resultate zeigen zudem, dass selbst bei einer geringen Anzahl an Trainingsdaten die erstellten Class-Embeddings zwar nicht perfekt positioniert sind, aber dennoch hinreichend repräsentativ für ihre jeweilige Klasse sind, um bei der Selektion von neuen Datenpunkten eine höhere Trefferanzahl zu erzielen als bei der Auswahl mittels Pretrained-BERT oder zufälliger Auswahl.

6.3 Entropie der Klassenverteilung

In Abbildung 5.5 wird der Verlauf der Entropie der Klassenverteilung im Trainingsdatensatz in Abhängigkeit von dessen Größe gezeigt. Wie bereits in Abschnitt 4.1.3 erläutert, wird die Entropie der Verteilung gemessen, um ihre Gleichmäßigkeit zu bestimmen.

Mit zunehmender Größe des Datensatzes nimmt die Entropie bei der zufälligen Datenselektion kontinuierlich ab, was auf eine unausgewogene Verteilung der Klassen hindeutet. Dieser Effekt ist bei einem Datensatz mit fünf häufigen und fünf seltenen Klassen erwartbar und erklärt sich durch die zufällige Selektion von Trainingsdaten. Mit steigender Größe des Trainingsdatensatzes nähert sich die Klassenverteilung im Trainingsdatensatz der Verteilung des gesamten Datensatzes an. Folglich sinkt die Entropie der Klassenverteilung im Trainingsdatensatz und tendiert gegen den Entropiewert des gesamten Datensatzes.

Die Entropie sinkt bei der Pretrained-BERT Methode im Vergleich zu den anderen Methoden am stärksten. In Abbildung 5.7 ist ersichtlich, dass zu Beginn des Datenselektionsprozesses eine Klasse bevorzugt wird, was zu einem rapiden Abfall der Entropie führt. In Abbildung 5.8 und Abbildung 5.9 ist zu erkennen, dass die Methode zwischen 300 und 650 Datenpunkten beginnt, eine weitere Klasse auszuwählen, was zu einem kurzfristigen Anstieg der Entropie führt, bevor sie aufgrund der unausgewogenen Klassenverteilung wieder sinkt. Insgesamt weist diese Methode den niedrigsten Entropiewert auf, was auf die ungleichmäßige Verteilung der Klassen im Datensatz zurückzuführen ist. Möglicherweise liegt dies daran, dass die Datenselektion bei dieser Methode nicht optimal durchgeführt wird. Wahrscheinlich liegt der Grund darin, dass die erstellten Class-Embeddings für ihre jeweilige Klasse nicht repräsentativ genug sind. Das liegt vermutlich an dem fehlenden Fine-Tuning und der Tatsache, dass das Vortraining des Sprachmodells nicht mit Gesetzestexten stattgefunden hat.

Die höchsten Entropiewerte liefern, wie auch die Ergebnisse der Treffer andeuten, die Datensätze, bei denen die Datenselektion auf fine-tuned BERT-Modellen basieren. Zu Beginn sinkt jedoch die Entropie bei allen Strategien, was darauf hindeutet, dass die Modelle noch nicht genügend mit den Daten vertraut sind und die Datenselektion noch nicht optimal ist. Abbildung 5.3 bestätigt diese Annahme, da man sieht, dass zu Beginn die Trefferanzahl geringer ist als im restlichen Verlauf des Prozesses. Für alle Methoden gilt, dass ab 200 Datenpunkten im Trainingsdatensatz die Trefferanzahl im Datenselektionsschritt steigt und das bewirkt, dass die Entropie steigt und sich dem theoretisch maximalen Wert (bei 10 Klassen etwa 3.3219) nähert. Das bedeutet, dass die Class-Embeddings mit einem Datensatz ab 200 Datenpunkten genug Daten haben, um als Repräsentanten für ihre Klasse zu fungieren. Sie beinhalten genug Informationen über die jeweilige Klasse, und sind dadurch in der Lage den unausgeglichenen Datensatz zu "reparieren". So wird eine Gleichverteilung der Klassen angenähert.

Ähnlich wie in Abschnitt 6.2 kann man auch hier zu dem Schluss kommen, dass ein Lernprozess stattfindet, bei dem die BERT-Instanz des Textklassifizierers ein Verständnis über die Trainingsdaten aufbaut und dann imstande ist, performante Class-Embeddings zu generieren.

6.4 Verteilung der Klassen

In Abbildung 5.6 bis Abbildung 5.9 sind die Verteilungen der Klassen, die durch die Methoden ”Random Selection”, ”Pretrained BERT” und ”Sum 12 Layers” entstehen, dargestellt.

Wir wissen bereits, dass jedes Experiment mit demselben initialen Trainingsdatensatz beginnt. Nur die unterschiedliche Datenselektion jeder Methode hat einen Einfluss darauf, wie sich der Trainingsdatensatz zusammensetzt und somit auch seine Klassenverteilung. Daraus folgt, dass die Abbildungen 5.6 bis 5.9 die unmittelbaren Effekte der unterschiedlichen Datenselektion abbilden. Wenn die Datenselektion viele Treffer erzielt, hat das zur Folge, dass diejenigen Datenpunkte ausgewählt werden, die zu einer Gleicherteilung der Klassen führen. Wenn hingegen wenige Treffer erzielt werden, kann das dazu führen, dass der Datensatz unausgeglichen wird, weil dann Datenpunkte ausgewählt werden, die nicht zu einer Gleichverteilung führen und somit die Unausgeglichenheit des Datensatzes verstärken.

Die gemessene Entropie in Abbildung 5.5 gibt Aufschluss über die Klassenverteilung, und somit hängen die Abbildungen in diesem Abschnitt und Abbildung 5.5 eng miteinander zusammen. Je höher der Entropiewert, desto gleichverteilter sind die Klassen im Datensatz, und je niedriger der Wert, desto unausgeglichener sind dann die Klassen.

Die Ergebnisse in Abbildung 5.6 bis Abbildung 5.9 verdeutlichen somit die Ergebnisse und Erkenntnisse von Abschnitt 6.2 und Abschnitt 6.3.

In den Abbildungen Abbildung 5.6 bis Abbildung 5.9 wird die hohe Performance der Class-Embeddings der ”Sum 12 Layers” Methode verdeutlicht. Dies lässt sich schon in Abbildung 5.5 ableiten, da bei dieser Methode die Entropie-Kurve nahezu kontinuierlich nahe am Maximum verläuft. Außerdem erzielt diese Methode ab einer Datensatzgröße von 250 konstant 48 oder mehr Treffer pro Active Learning Runde.

Die Resultate der Pretrained-BERT Methode liefern eine Erklärung für das wellenartige Verhalten der Entropie-Kurve in Abbildung 5.5. Zu Beginn des Prozesses wird die Klasse ”export refund” (3568) bevorzugt. Dies führt zu einem rapiden Abfall des Entropiewertes. Daraufhin wählt die Methode häufig die Klasse ”ship’s flag” (2282) aus, was den kurzzeitigen Anstieg des Entropiewertes erklärt. Das erneute Absinken des Entropiewerts am Ende kommt davon, dass weiterhin Datenpunkten aus den beiden Klassen ”export refund” (3568) und ”ship’s flag”(2282) bei der Datenselektion bevorzugt werden. Insgesamt zeigt sich, dass die Datenselektion bei diesem Ansatz unzureichend ist. Obwohl für die Erstellung der Class-Embeddings auch die ”Sum 12 Layers” Methode verwendet wird, wird das Modell nicht fine-tuned. Dies erklärt möglicherweise das Ergebnis, da BERT mit Daten aus dem Internet trainiert ist [DCLT18], darunter Wikipedia-Artikel, Webtexte und Bücher. Es ist daher wahrscheinlich, dass BERT nicht speziell auf europäische Gesetzestexte trainiert ist und kein Verständnis für diese Art von Texten besitzt. Aus diesem

Grund sind die vorhandenen Gewichtungen in BERT möglicherweise nicht geeignet für unseren Datensatz.

Mit zunehmender Größe des Trainingsdatensatzes wird in der zufallsbasierten Methode deutlich, wie sich der Gesamtdatensatz zusammensetzt. Bereits ab 350 Datenpunkten, von denen 250 zufällig ausgewählt sind, wird erkennbar, dass der Gesamtdatensatz aus 5 häufigen Klassen und 5 seltenen Klassen besteht. Mit zunehmender Anzahl von hinzugefügten Datenpunkten wird der Unterschied zwischen den häufigen und seltenen Klassen immer deutlicher. Das ist auch in Abbildung 5.5 zu sehen, weil die Entropie des Trainingsdatensatzes bei dieser Methode stetig sinkt. Dieses Ergebnis ist zu erwarten, da bei der zufälligen Auswahl von Datenpunkten lediglich eine Annäherung an die Klassenverteilung des Gesamtdatensatzes erfolgen kann.

6.5 Abstandsmatrix

In Abbildung 5.10 sind die resultierenden Abstände von den Class-Embeddings, die bei der ”Sum 12 Layers” Methode entstehen. Die Zeilen und Spaltenindizes sind die Klassen, und jeder Eintrag gibt den Abstand von den jeweiligen Indizes (Klassen) zueinander an. Um den semantischen Sinn der jeweiligen Klassen zu verstehen, sind in Tabelle 5.1 die jeweiligen Klassenbedeutungen angeführt. Für den Abstand wird die euklidische Norm verwendet und das Ergebnis nach dem Komma abgeschnitten, sodass nur ganze Zahlen resultieren.

6.5.1 Intuition

Bevor wir uns die Ergebnisse anschauen, möchten wir die vorkommenden Klassen genauer betrachten und intuitiv beschreiben, welche von ihnen vom Namen und der Bedeutung her semantische Ähnlichkeiten besitzen.

Klassen 1118 (pip fruit), 1605 (fruit vegetable) und 693 (citrus fruit) behandeln europäische Verordnungen für Früchte und Gemüse. Daher betrachten wir diese als semantische ähnlich.

Klassen 1309 (import), 3568 (export refund), 3591 (quantitative restrictions) und 1642 (export licence) behandeln Verordnungen über den Import und Export von Warengütern. Intuitiv passen 1309 und 3591 enger zusammen, da der Import von Waren oftmals quantitativen Restriktionen unterliegen. Die Klassen 3568 und 1642 passen auch enger zusammen, da diese sie sich auf den Export auf Waren beziehen.

Die Klassen 1318 (Germany) und 2563 (Portugal) beziehen sich auf Verordnungen für die jeweiligen Länder. Beides sind Klassen über europäische Länder, aber das muss nicht bedeuten, dass beide Länder denselben Verordnungen unterstehen. Deswegen sehen wir keine große semantische Ähnlichkeit zwischen ihnen. Da die Verordnungen unter anderem den Handel innerhalb der Europäischen Union reguliert, könnte eine Beziehung zu den Import und Export Klassen (1309, 3568, 3591, 1642) in den Daten vorhanden sein.

Die Klasse 2282 (ship’s flag) lässt sich intuitiv keiner der anderen Klasse zuordnen. Schiffsflaggen haben keine Ähnlichkeit zu Früchten und Gemüse. Sie können zwar gehandelt werden und unterliegen somit Import und Export Verordnungen, aber wir nehmen nicht an, dass es gesonderte Verordnungen für den Import und Export von Schiffsflaggen gibt. Schiffsflaggen repräsentieren ihr jeweiliges Land, aber wir nehmen auch nicht an, dass es spezielle Verordnungen für Schiffsflaggen aus Deutschland oder Portugal gibt. Somit können wir für diese Klasse keine große semantische Ähnlichkeit zu den anderen Klassen feststellen.

6.5.2 Tatsächliche Ergebnisse

Früchte und Gemüse - (1118, 1605, 693)

In Abbildung 5.10 sieht man, dass die Class-Embeddings für die Klassen 1118, 1605 und 693 am nächsten zueinander sind. Ihr Abstand beträgt 0 und 1. Das deckt sich mit der Intuition, die wir vorher hatten. Mit den restlichen Klassen gibt es die größten Abstände, die in der Matrix vorkommen.

Import und Export - (1309, 3568, 3591, 1642)

Bei den Import und Export Verordnungen zeichnet sich ein ähnliches Bild ab, wie wir es vorher vermutet haben. Zusätzlich trifft die Vermutung zu, dass die Klassen 1309 (import) und 3591 (quantitative restrictions) sowie die Klassen 3568 (export refund) und 1642 (export licence) innerhalb der Gruppe am nächsten zueinander sind. Außerdem kann man sehen, dass in manchen Fällen die Klassen alle eine ähnliche Distanz zu den Länderklassen haben, wie zum Beispiel Klasse 1309 (import).

Deutschland und Portugal - (1318, 2563)

Die beiden Klassen 1318 (Germany) und 2563 (Portugal) haben einen relativ mittleren Abstand zueinander. Der Abstand voneinander ist nicht größer als zum Beispiel zu den Klassen, die Verordnungen über Früchte und Gemüse abdecken, er ist jedoch größer als der Abstand der beiden Länder zu der Klasse 1309 (import). Zusätzlich ist ihr beider Abstand zu der Klasse 2282 (ship's flag) größer als zueinander, sodass man sagen kann, dass die Verordnungen wenig bis gar nicht ähnlich sind. Auch diese Ergebnisse wurden in der Intuition vermutet.

Schiffsflaggen - (2282)

Die Klasse 2282 (ship's flag) besitzt zu allen anderen Klassen eine mittel bis weite Distanz. Dies deckt sich mit unserer Intuition, dass die Verordnungen über Schiffsflaggen wenig bis gar keine Ähnlichkeiten zu den anderen Verordnungen haben. Schaut man sich die Verordnungen an, dann kommt man zu dem Schluss, dass die meisten von ihnen die Befugnisse in der Fischerei regulieren, wenn man unter bestimmten Schiffsflaggen fischt. Man sieht also, dass semantisch keine Ähnlichkeit zu den anderen Klassen besteht, was sich auf die Distanzen zu den anderen Class-Embeddings auswirkt.

Zusammenfassung

Die Ergebnisse zeigen, dass alle unsere Vermutungen über die semantische Ähnlichkeit zwischen den Klassen zutreffen.

Die semantische Ähnlichkeit zwischen unterschiedlichen Klassen schlägt sich in den Ergebnissen der Abstandsmatrix nieder, denn semantisch ähnliche Klassen haben Class-Embeddings, die nah beieinander sind und semantisch unähnliche Klassen haben Class-Embeddings, die weit voneinander entfernt sind.

Des Weiteren zeigt sich, dass die erzeugten Class-Embeddings im Vektorraum so positioniert sind, dass ihr Abstand zueinander den Grad ihrer Ähnlichkeit widerspiegelt. Das bedeutet, dass eine Klasse, die beispielsweise zwei anderen Klassen ähnlich ist, der Klasse näher liegt, die semantisch ähnlicher ist.

Diese Eigenschaft liefert wertvolle Informationen über den vorliegenden Datensatz und dessen vorkommenden Klassen.

Beispiele für die Verwendung der gelieferten Informationen:

1. **Semantische Ähnlichkeit:** Die Positionierung der Class-Embeddings im Vektorraum ermöglicht es, semantische Ähnlichkeiten zwischen Klassen zu erfassen. Dadurch können Klassen identifiziert werden, die ähnliche Merkmale oder Eigenschaften teilen.
2. **Klassifikation und Clustering:** Die Ähnlichkeiten zwischen den Class-Embeddings können für Klassifikations- und Clustering-Aufgaben genutzt werden. Klassen, die nahe beieinander liegen, können als zusammengehörig betrachtet oder in ähnliche Gruppen gruppiert werden.
3. **Visualisierung:** Durch die räumliche Positionierung der Class-Embeddings im Vektorraum können sie visuell dargestellt werden. Dies ermöglicht eine intuitive Visualisierung von Klassenähnlichkeiten und -beziehungen, was die Interpretierbarkeit des Systems verbessert.
4. **Effiziente Suche:** Die ähnlichkeitsbasierte Positionierung der Class-Embeddings ermöglicht eine effiziente Suche nach ähnlichen Klassen. Dies kann beispielsweise für die Information Retrieval oder das Finden von ähnlichen Klassen in großen Datensätzen nützlich sein.

7 Fazit

In dieser Arbeit haben wir die Erstellung von Embeddings und somit auch die Erstellung von Class-Embeddings (Zentroid aller Embeddings einer Klasse) durch das Sprachmodell *BERT* untersucht. Um untersuchen zu können, wie repräsentativ die erstellten Class-Embeddings für ihre jeweilige Klasse sind, wurden Experimente in einem Active Learning Szenario durchgeführt. In dem Active Learning Prozess wurde untersucht, ob man die Klasse eines neuen und ungelabelten Embeddings anhand dessen Nähe zu den Class-Embeddings im Vektorraum vorherbestimmen kann.

Durch das gewählte Szenario waren wir in der Lage zu ermitteln, wie sich die Performance der Class-Embeddings in Abhängigkeit von der Trainingsdatensatzgröße entwickelt und wie sie mit neuen unbekannten Daten umgehen.

Die Evaluation unserer Ergebnisse zeigt, dass die generierten Class-Embeddings effektiv genutzt werden können, um die Klassen von ungelabelten Datenpunkten vorherzusagen (siehe Abschnitt 6.2). Die vorhergesehene Klasse des Datenpunktes leitet sich von dem nächstgelegenen Class-Embedding ab.

Dadurch können Class-Embeddings zum Beispiel im Active Learning Prozess verwendet werden, um eine Gleichverteilung von Klassen in einem Datensatz zu erzielen, auch wenn dessen Datenquelle eine ungleichverteilte Klassenverteilung wiedergibt (siehe Abschnitt 6.3 bis Abschnitt 6.4).

Außerdem haben wir gezeigt, dass sich die intuitive semantische Ähnlichkeit von den jeweiligen Klassen zueinander von den Abständen der jeweiligen Class-Embeddings ableiten lässt (siehe Abschnitt 6.5). Die erzeugten Class-Embeddings sind so im Vektorraum positioniert, dass ihr Abstand zueinander den Grad ihrer Ähnlichkeit widerspiegelt.

Eine Nebenerkenntnis unserer Arbeit ist, dass die durch die Class-Embeddings erstellten Trainingsdatensätze im Vergleich zu den zufällig zusammengestellten Trainingsdatensatz keine signifikante Steigerung der Performance des Textklassifizierers herbeiführen (siehe Abschnitt 6.1). Dies ist nicht auf eine mindere Performance der Class-Embeddings zurückzuführen, denn diese haben das Ziel eines gleichverteilten Datensatzes erreicht. Wir gehen davon aus, dass der Textklassifizierer auch im zufälligen Szenario ausreichend Daten für jede Klasse erhält, um äquivalente Ergebnisse im Vergleich zu den Embedding-basierten Strategien zu erzielen.

Die erzielten Ergebnisse und Erkenntnisse lassen darauf schließen, dass das Sprachmodell BERT sehr effektiv bei der Generierung von Embeddings und Class-Embeddings ist, da es als Grundlage für deren Erstellung diente.

Es sei erwähnt, dass die hier vorgestellten Experimente lediglich auf einem Datensatz durchgeführt wurden. Um eine bessere Sicht auf die allgemeine Leistung der Class-Embeddings zu erhalten, ist es sinnvoll, dieselben Experimente auch auf anderen Datensätzen durchzuführen und zu untersuchen, wie die Class-Embeddings sich dann verhalten.

Beispielsweise könnte man einen Datensatz verwenden, welcher eine viel höhere Anzahl an Klassen beinhaltet, um die Repräsentanz von Class-Embeddings zu untersuchen. Mit einem Datensatz, dessen Klassen semantisch ähnlich sind, wäre man in der Lage zu untersuchen, ob die Class-Embeddings diese unterscheiden können.

Generell lohnt es sich Datensätze aus anderen Sprachdomänen auszuwählen und dieselben Experimente darauf laufen zu lassen, um verifizieren zu können, dass die in dieser Arbeit erbrachten Ergebnisse kein Zufall sind, sondern auf die allgemeine Robustheit und Übertragbarkeit der BERT-basierten Class-Embeddings hinweisen.

Wenn sich die Ergebnisse bei anderen Datensätzen wiederholen, ergeben sich damit verschiedene Möglichkeiten im Bereich der natürlichen Sprachverarbeitung. Class-Embeddings können verwendet werden, um erste Informationen über einen noch unbekannten Datensatz zu erhalten, zum Beispiel wie die vorkommenden Klassen semantisch zueinander stehen. Auch können sie verwendet werden, wenn Daten vorhanden sind, die wenig gelabelt sind und man einen annotierten Datensatz zusammenstellen möchte, dessen Klassen gleichverteilt sind. Das sind nur einige Beispiele, die sich mit Class-Embeddings ergeben.

Insgesamt erfüllen Class-Embeddings das angestrebte Ziel der Repäsentanz einzelner Klassen. Weiterführende Experimente mit unterschiedlichen Datensätzen werden die Robustheit und Präzision weiter untersuchen.

Literaturverzeichnis

[AWH18] B. An, W. Wu, H. Han. „Deep Active Learning for Text Classification“. In: *Proceedings of the 2nd International Conference on Vision, Image and Signal Processing*. ICVISP 2018. Las Vegas, NV, USA: Association for Computing Machinery, 2018. ISBN: 9781450365291. DOI: 10.1145/3271553.3271578. URL: https://doi.org/10.1145/3271553.3271578 (zitiert auf S. 12).

[BCB14] D. Bahdanau, K. Cho, Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473 (zitiert auf S. 15).

[BU16] M.-F. Balcan, R. Urner. „Active Learning – Modern Learning Theory“. In: *Encyclopedia of Algorithms*. Hrsg. von M.-Y. Kao. New York, NY: Springer New York, 2016, S. 8–13. ISBN: 978-1-4939-2864-4. DOI: 10.1007/978-1-4939-2864-4_769. URL: https://doi.org/10.1007/978-1-4939-2864-4_769 (zitiert auf S. 18).

[CHU17] K. W. CHURCH. „Word2Vec“. In: *Natural Language Engineering* 23.1 (2017), S. 155–162. DOI: 10.1017/S1351324916000334 (zitiert auf S. 13).

[DCLT18] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805 (zitiert auf S. 12, 13, 15–18, 20, 22, 52).

[EHG+20] L. Ein-Dor, A. Halfon, A. Gera, E. Shnarch, L. Dankin, L. Choshen, M. Danilevsky, R. Aharonov, Y. Katz, N. Slonim. „Active Learning for BERT: An Empirical Study“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, S. 7949–7962. DOI: 10.18653/v1/2020.emnlp-main.638. URL: https://aclanthology.org/2020.emnlp-main.638 (zitiert auf S. 12).

[GKBG16] M. Goudjil, M. Koudil, M. Bedda, N. Ghoggali. „A Novel Active Learning Method Using SVM for Text Classification“. In: *International Journal of Automation and Computing* 15 (Juli 2016). DOI: 10.1007/s11633-015-0912-z (zitiert auf S. 12).

[GLT21] A. Galassi, M. Lippi, P. Torroni. „Attention in Natural Language Processing“. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10 (2021), S. 4291–4308. DOI: 10.1109/TNNLS.2020.3019893 (zitiert auf S. 15).

[GVV98] A. Gammerman, V. Vovk, V. Vapnik. „Learning by Transduction“. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. UAI'98. Madison, Wisconsin: Morgan Kaufmann Publishers Inc., 1998, S. 148–155. ISBN: 155860555X (zitiert auf S. 20).

[HG09] H. He, E. A. Garcia. „Learning from Imbalanced Data“. In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), S. 1263–1284. DOI: 10.1109/TKDE.2008.239 (zitiert auf S. 19).

[LG94] D. D. Lewis, W. A. Gale. „A Sequential Algorithm for Training Text Classifiers“. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '94. Dublin, Ireland: Springer-Verlag, 1994, S. 3–12. ISBN: 038719889X (zitiert auf S. 20).

[MSD12] L. E. Makili, J. A. V. Sánchez, S. Dormido-Canto. „Active Learning Using Conformal Predictors: Application to Image Classification“. In: *Fusion Science and Technology* 62.2 (2012), S. 347–355. DOI: 10.13182/FST12-A14626. URL: https://doi.org/10.13182/FST12-A14626 (zitiert auf S. 20).

[PMM21] S. Prabhu, M. Mohamed, H. Misra. „Multi-class Text Classification using BERT-based Active Learning“. In: (2021). DOI: 10.48550/ARXIV.2104.14289. URL: https://arxiv.org/abs/2104.14289 (zitiert auf S. 12).

[PSM14] J. Pennington, R. Socher, C. Manning. „GloVe: Global Vectors for Word Representation“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Okt. 2014, S. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (zitiert auf S. 13).

[RG19] N. Reimers, I. Gurevych. „Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, S. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410 (zitiert auf S. 13).

[Set09] B. Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009 (zitiert auf S. 18, 20).

[Sha48] C. E. Shannon. „A mathematical theory of communication“. In: *The Bell System Technical Journal* 27.3 (1948), S. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x (zitiert auf S. 28).

[SN20] C. Schröder, A. Niekler. *A Survey of Active Learning for Text Classification using Deep Neural Networks*. 2020. DOI: 10.48550/ARXIV.2008.07267. URL: https://arxiv.org/abs/2008.07267 (zitiert auf S. 12).

[SOS92] H. S. Seung, M. Opper, H. Sompolinsky. „Query by Committee“. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, S. 287–294. ISBN: 089791497X. DOI: 10.1145/130385.130417. URL: https://doi.org/10.1145/130385.130417 (zitiert auf S. 20).

[SQXH19] C. Sun, X. Qiu, Y. Xu, X. Huang. *How to Fine-Tune BERT for Text Classification?* 2019. DOI: 10.48550/ARXIV.1905.05583. URL: https://arxiv.org/abs/1905.05583 (zitiert auf S. 15).

[TDP19] I. Tenney, D. Das, E. Pavlick. „BERT Rediscovers the Classical NLP Pipeline“. In: (2019). DOI: 10.48550/ARXIV.1905.05950. URL: https://arxiv.org/abs/1905.05950 (zitiert auf S. 17, 20).

[TK01] S. Tong, D. Koller. „Support Vector Machine Active Learning With Applications To Text Classification“. In: *The Journal of Machine Learning Research* 2 (Dez. 2001), S. 45–66 (zitiert auf S. 12).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (zitiert auf S. 15, 16).

[WBMK22] L. Wertz, J. Bogojeska, K. Mirylenka, J. Kuhn. „Evaluating Pre-Trained Sentence-BERT with Class Embeddings in Active Learning for Multi-Label Text Classification“. In: *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Online only: Association for Computational Linguistics, Nov. 2022, S. 366–372. URL: https://aclanthology.org/2022.aacl-short.45 (zitiert auf S. 13, 19, 30, 48).

[WMKB22] L. Wertz, K. Mirylenka, J. Kuhn, J. Bogojeska. „Investigating Active Learning Sampling Strategies for Extreme Multi Label Text Classification“. In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, Juni 2022, S. 4597–4605. URL: https://aclanthology.org/2022.lrec-1.490 (zitiert auf S. 12).

[YDL+18] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, Q. V. Le. *QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension*. 2018. DOI: 10.48550/ARXIV.1804.09541. URL: https://arxiv.org/abs/1804.09541 (zitiert auf S. 15).

Alle URLs wurden zuletzt am 31. 05. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift