Universität Leipzig
Institut für Informatik
Studiengang Informatik, B.Sc.

Adapting sentence embeddings to OCR erroneous data

Bachelorarbeit

Jonas Stahl
geb. am: 12.01.1995 in Freiburg im Breisgau

Matrikelnummer 3743881

1. Gutachter: Dr. Christian Kahmann

Datum der Abgabe: 23. Juni 2023

Erklärung

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

Leipzig, 23. Juni 2023

Jonas Stahl

Inhaltsverzeichnis

1 Abstract **1**

2 Einleitung **2**

3 Verwandte Arbeiten **3**
3.1 Word2Vec 3
3.2 BERT 4
3.3 Sentence-BERT 6
3.4 Augmented SBERT 6
3.5 Optical Character Recognition 8
3.6 OCR Error Correction 8

4 Eigener Ansatz **9**

5 Experimente **12**
5.1 Kosinus-Ähnlichkeit 13
5.1.1 Datensätze 13
5.2 Textklassifizierung 15
5.2.1 Textklassifizierung *IMDB* 15
5.2.2 Textklassifizierung *Hatespeech* 17
5.2.3 Nähere Betrachtung Modell s5 18

6 Fazit **19**

Literaturverzeichnis **21**

Kapitel 1

Abstract

In dieser Arbeit wird eine neue Herangehensweise vorgestellt, um die Performance von SBERT-Modellen bei der Verarbeitung von Texten mit OCR-Fehlern zu verbessern. Dafür werden SBERT-Modelle trainiert, deren Trainingsdaten auf unterschiedliche Weise gezielt mit OCR-Fehlern versehen werden. Anschließend werden die Modelle anhand verschiedener Szenarien getestet. Die Ergebnisse der Experimente zeigen je nach Trainingsansatz eine positiven sowie negativen Effekt auf die Performanz der Modelle. Die Szenarien, bei welchen die Performanz eines SBERT-Modells gesteigert werden konnte, deuten auf ein grundsätzliches Funktionieren der Herangehensweise hin.

Kapitel 2

Einleitung

Digitaler Text existiert frühestens seit der Entwicklung des Computers in den 40er Jahren, wobei zu erwarten ist, dass nennenswerte Mengen erst seit der Verbreitung von privat genutzten Computern entstehen und die Generierung durch die weltweite Nutzung des Internets ab 1990 exponentiell steigt. HistorikerInnen nehmen an, dass der Mensch spätestens 3.000 Jahre vor Christus mit dem Schreiben [8], also dem Generieren von analogem Text, begonnen hat. Auch wenn es unmöglich ist, die genaue Menge von nicht-digitalisiertem Text festzustellen, kann von einer signifikanten Größe ausgegangen werden. Für GeisteswissenschaftlerInnen aller Art sind diese älteren Schriften keineswegs uninteressant, doch durch die analoge Beschaffenheit der Daten gestaltet sich deren Verarbeitung mühsam und zeitaufwändig.

Die Optische Zeichenerkennung (OCR) ist ein fester Bestandteil der modernen natürlichen Sprachverarbeitung. Um mit großen Mengen von analogem Text wissenschaftlich sinnvoll und effizient arbeiten zu können, werden moderne digitale Technologien verwendet, welche in der Regel auf einer Variation des maschinellen Lernens basieren. Für die Verwendung solcher Technologien ist eine digitale Erfassung und die damit erreichte Maschinenlesbarkeit der Schriften unerlässlich. Obwohl die Technik zur maschinellen Identifizierung von Zeichen im 21. Jahrhundert stetig weiterentwickelt und verbessert wird, bestehen abhängig von der Beschaffenheit und Qualität der analogen Vorlage nach wie vor teilweise hohe Fehlerraten [5]. Typische Fehler entstehen durch die Verwechslung von optisch ähnlichen Zeichen. Dabei wird beispielsweise der Buchstabe s als Ziffer 5 eingelesen. Wo bei von einer Maschine gedrucktem Text eine Fehlerrate von bis zu 10% als gut angesehen werden kann [3], muss man bei beispielsweise kursiver Handschrift von einer weit höheren Fehlerrate ausgehen. Werden solche fehlerhaften Daten zur weiteren Verarbeitung genutzt, um beispielsweise eine Sentimentanalyse durchzuführen, können inkorrekte Zeichen deren Resultate beeinflussen oder sogar unbrauchbar machen.

Kapitel 3

Verwandte Arbeiten

3.1 Word2Vec

Durch die stetig wachsende Bedeutung von künstlicher Intelligenz und dem Maschinellen Lernen wird das dementsprechende Forschungsfeld immer vielfältiger und größer. Wo neue Technologien erarbeitet werden, entstehen in der Regel auch Hürden, die es zu überwinden gilt. Die Worteinbettungstechnik Word2vec [6][7] wurde 2013 veröffentlicht und galt lange als grundlegend. Word2vec zielt darauf ab, einzelne Wörter als Vektor abzubilden - also sogenannte Worteinbettungen zu generieren - und letztendlich mögliche Beziehungen zwischen verschiedenen Wörtern zu erkennen. Anhand Tabelle 3.2 können wir beispielhaft den Vektor (0.6, 14, 1) für das Wort *Gurke* ablesen. Außerdem ist das Modell anhand der Vektoren in der Lage, Beziehungen oder Ähnlichkeiten zwischen den Wörtern zu erkennen. Die Anfrage *Hühnerschenkel+Vegan* würde als Ergebnis *Tofu* liefern, da zum Einen die Eigenschaft Vegan erfüllt ist und zum Anderen die Werte für Protein und Kalorien sehr ähnlich sind. Die in 3.2 dargestellte Tabelle ist natürlich stark vereinfacht - in der Praxis nutzt Word2Vec einen 300-dimensionalen Vektorraum. Die zwei Hauptmethoden des Trainings von Word2Vec sind *Continuous Bag of Words* (CBOW) und *Skip-gram*. Das Hauptziel bei CBOW besteht darin, ein Zielwort basierend auf den umliegenden Kontextwörtern vorherzusagen. Für das Training werden Tupel generiert, welche aus Kontextwörtern als Input und dem Zielwort als Output bestehen und zur Identifizierung im Wortschatz als One-Hot Vektor umgewandelt werden. One-Hot Vektoren sind Bitfolgen, welche nur eine 1 enthalten (siehe Tabelle 3.1). Die durch das Multiplizieren der Vektoren mit einer Gewichtsmatrix erhaltenen Worteinbettungen werden anschließend in ein Neurales Netzwerk eingespeist, welches durch diese in der Lage ist, die implizite Beziehung zwischen den Wörtern zu erlernen. Das Neurale Netzwerk wird schlussendlich darauf trainiert, das Zielwort anhand der Kon-

textwörter zu erkennen. Der Trainingsprozess bei Skip-gram ähnelt dem des CBOW stark, wobei das Ziel bei Skip-gram darin besteht, die Kontextwörter basierend auf einem Zielwort zu erkennen. Ein großer Nachteil von Word2Vec ist, dass es für ein einzelnes Wort nur einen einzigen Vektor geben kann. Dabei wird nicht beachtet, dass Wörter - mindestens in der deutschen Sprache - mehrere unterschiedliche Bedeutungen haben können. Das deutsche Polysem *Bank* hat beispielsweise zwei zueinander beziehungslose Bedeutungen, welche bei der mathematischen Verarbeitung von Word2vec nicht beachtet werden. Wird nun eine hypothetische Anfrage basierend auf der *(Park-)bank* gestellt, der Vektor für das Wort *Bank* stellt allerdings die *(Finanz-)bank* dar, wird das Ergebnis unbrauchbar. Da es für diese Problematik in Word2Vec keine Lösung gibt, in die Arbeit mit dieser Technik nicht zuverlässig.

Dezimal	Binär	One-Hot
0	00000	00001
1	00001	00010
2	00010	00100
3	00011	01000
4	00100	10000

Tabelle 3.1: Beispiel für One-Hot Vektoren

	Protein (g/100g)	Kalorien (kcal/100g)	Vegan
Tofu	17	163	1
Gurke	0.6	14	1
Kuhmilch	3.4	272	0
Mehl	10	348	1
Butter	0.6	741	0
Hühnerschenkel	18	174	0

Tabelle 3.2: Beispiel für Eigenschaftsvektoren bei Word2Vec

3.2 BERT

Das 2019 von Google entwickelte Sprachmodell *Bidirectional Encoder Representations from Transformers* [2], kurz *BERT*, bietet eine bis dato neue Herangehensweise in der Natürlichen Sprachverarbeitung. BERT betrachtet beim

Generieren einer Worteinbettung ebenfalls die umliegenden Wörter und erkennt gegebenenfalls die dynamische Semantik des zu verarbeitenden Wortes. Wo Word2vec bei dementsprechenden Sätzen den Unterschied zwischen der Bank als Finanzgebäude und der Bank als Sitzmöglichkeit nicht erkennen würde, ist BERT im Idealfall in der Lage, diesbezüglich zu unterscheiden. Seit 2022 gilt Word2vec daher als veraltet, während Transformer Modelle wie BERT den aktuellen Stand der Technik abbilden. Während herkömmliche Sprachmodelle Text von links nach rechts oder umgekehrt verarbeiten, ist BERT seinem Namen entsprechend ein bidirektionales Modell, wodurch es sowohl den linken als auch den rechten Kontext eines Wortes in einem Satz kennt. BERT basiert auf einem von Mechanismen der Selbstaufmerksamkeit gestützten neuralen Netzmodell. Unter Selbstaufmerksamkeit versteht man die Fähigkeit, zwischen Informationen mit höherer Wichtigkeit und Informationen mit niedrigerer Wichtigkeit unterscheiden zu können. Stellt man sich ein Bild vor, welches den Himmel sowie in einem kleinen Teil des Bildes einen einzelnen Vogel abbildet und in mehrere Quadrate aufgeteilt ist, könnte man dem Quadrat, welches den Vogel enthält, eine höhere Wichtigkeit oder Bedeutung zuordnen. BERT ist in der Lage, diese Abwägung bei den verschiedenen Wörtern eines Satzes zu treffen.

Das Training eines BERT-Modells besteht aus zwei Phasen: dem Vortraining *(Pretraining)* und der Feinabstimmung *(Fine-Tuning)*. Während des Vortrainings, welches maßgeblich aus Masked Language Modeling (MLM) und Next Sentence Prediction (NSP) besteht, wird sozusagen das Grundgerüst des Modells gebaut. Ähnlich wie bei Word2Vec werden Worteinbettungen generiert, wobei BERT bei deren Erstellung ein gewissen kontextuelles Verständnis in Bezug auf den ganzen Satz, in welchem ein Wort enthalten ist, einfängt. Mithilfe mehrere Schichten von Selbstaufmerksamkeitsmechanismen und Neurale *Feed-Forward* Netzwerke ist BERT ebenfalls in der Lage, verschiedene Bedeutungen eines einzelnen Wortes zu erkennen. Bei MLM versucht das Modell, maskierte Wörter eines Satzes vorherzusagen. Anhand des Originalsatzes lernt BERT, den Kontext der unmaskierten Wörter zu verstehen. Bei NSP liest BERT zwei Sätze ein und versucht vorherzusagen, ob der zweite Satz des Paares tatsächlich auf den ersten Satz folgt oder ein zufälliger, anderer Satz ist. Hierdurch lernt das Modell, eventuelle Beziehungen zwischen Sätzen zu erkennen. Ist das Vortraining abgeschlossen, wird das Modell noch auf eine spezifische Aufgabe, beispielsweise eine Sentimentanalyse oder eine Fragenbeantwortung feinabgestimmt.

3.3 Sentence-BERT

Die ebenfalls 2019 veröffentlichte Arbeit *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* [9], kurz *SBERT*, baut auf BERT auf und zielt darauf ab, Satzeinbettungen bzw. Vektorrepräsentationen fester Länge für einzelne Sätze zu erstellen und anhand dieser Sätze zu vergleichen. Für das Training eines SBERT-Modells erhält es Satzpaare mit dazugehörigen Werten für die semantische Ähnlichkeit der beiden Sätze und erzeugt mittels Bi-Encoder eine Satzeinbettung für beide Sätze. Dabei wird BERT genutzt, um für jedes Wort eines Satzes eine Worteinbettung zu generieren und die Menge der resultierenden Einbettungen zusammengebracht (Pooling). Schlussendlich entsteht eine Repräsentation des Satzes - die Satzeinbettung. Ist das Training des SBERT-Modells abgeschlossen, ist es in der Lage selbstständig Satzeinbettungen für neue Sätze zu generieren. Diese Ähnlichkeitswerte der Satzeinbettungen können effizient ermittelt werden, um typische NLP-Aufgaben wie beispielsweise eine Textklassifizierung oder Clustering durchzuführen. Laut den beiden AutorInnen der TU Darmstadt benötigt BERT für die Suche nach dem Satzpaar mit der höchsten semantischen Ähnlichkeit in einer Sammlung aus 10.000 Sätzen circa 65 Stunden. Die Nutzung von SBERT reduziert den zeitlichen Aufwand des genannten, beispielhaften Szenarios bei gleichbleibender Genauigkeit von circa 65 Stunden auf 5 Sekunden.

3.4 Augmented SBERT

Für die Berechnung von Satzpaar-Repräsentationen im Vektorraum existieren zwei Herangehensweisen: Cross-Encoder, welche beispielsweise bei BERT eingesetzt werden und die von SBERT verwendeten Bi-Encoder. Cross-Encoder verarbeiten ein Satzpaar als einen vereinten Datenpunkt, sie erzeugen also einen einzelnen Output für zwei Sätze und schaffen es dabei, die Beziehung zwischen beiden Sätzen einzufangen. Der Output des Cross-Encoders beschreibt die Ähnlichkeit der beiden Sätze mit einem Wert zwischen 0 und 1. Bi-Encoder hingegen berechnen für beide Sätze eines Satzpaars eine Satzeinbettung. Die daraus entstehenden Repräsentationen im Vektorraum lassen sich bequem mittels Kosinus-Ähnlichkeit vergleichen. Während Cross-Encoder im Normalfall eine bessere Performanz erzielen, sind sie für einige Anwendungen der Natürlichen Sprachverarbeitung schlichtweg zu langsam [10]. Bi-Encoder hingegen benötigen viele Trainingsdaten sowie aufwändige Feinabstimmung, um eine gleichwertige Performanz zu erreichen.

Die AutorInnen der Arbeit *Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks* [10] stel-

len eine Strategie zur Anreicherung von Trainingsdaten vor, um Bi-Encoder effizienter trainieren zu können. Grundsätzlich besteht die Strategie daraus, ungekennzeichnete Satzpaare mit einem Cross-Encoder zu kennzeichnen. Der dabei entstehende Silber Datensatz wird mit dem Basis Datensatz - hier genannt Gold Datensatz - vereint, um die gesamte Menge an Trainingsdaten zu erhöhen. Dieser erweiterte Datensatz wird letztendlich genutzt, um den Bi-Encoder zu trainieren. Bei der Auswahl der ungekennzeichneten Satzpaare wenden die Wissenschaftler der TU Darmstadt verschiedene Strategien an[10].

Beim *Random Sampling* (RS) werden zufällige Sätze ausgewählt. Da hierbei die Wahrscheinlichkeit extrem gering ist, zwei Sätze mit hoher Ähnlichkeit auszuwählen, tendiert der resultierende Silber Datensatz zu einer ungleichmäßigen Verteilung in Bezug auf negative oder positive Satzpaare. Um entweder einen Datensatz mit gleichmäßiger Verteilung der Kennzeichnung oder gezieltem Schwerpunkt auf positiv gekennzeichneten Satzpaaren, beispielsweise für eine Klassifizierung, zu erhalten, kann die Strategie *Kernel Density Estimation* (KDE) verwendet werden. Hierbei werden ebenfalls zufällig ausgewählte Satzpaare gekennzeichnet. Im nächsten Schritt werden nur die Satzpaare behalten, welche eine der Aufgabe entsprechend gewünschte Kennzeichnung besitzen. Was die Rechenleistung betrifft, ist diese Herangehensweise eher ineffizient, weil viele der berechneten Kennzeichen im Endeffekt nicht genutzt werden. Beim *BM25 Sampling* (BM25) wird für jeden einzigartigen Satz des Gold Datensatzes mittels ElasticSearch[1] eine Menge der Größe k der ähnlichsten Sätze ermittelt und die resultierenden Satzpaare per Cross-Encoder gekennzeichnet. BM25 ist zwar recht effizient, allerdings können so nur Sätze gekennzeichnet werden, welche ein gewisses Vorkommen im Sprachgebrauch aufweisen. Um dies zu umgehen, wird im Ansatz *Semantic Search Sampling* (SS) der Bi-Encoder genutzt, um mittels Kosinus-Ähnlichkeit ähnliche ungekennzeichnete Satzpaare im Gold Datensatz zu finden und zu kennzeichnen. In der letzten vorgestellten Strategie (BM25-S.S.) wird das BM25 Sampling und das Semantic Search Sampling verknüpft, um deren Vorteile zu vereinen.

Beim Testen der Performanz von BERT, SBERT sowie AugmentedSBERT in Bezug auf die verschiedenen Sampling Strategien anhand Regressions- und Klassifizierungsaufgaben erreicht AugmentedSBERT eine höhere Performanz im Vergleich zu SBERT. Im Vergleich der Sampling Strategien erzielt die *Kernel Density Estimation* die besten Werte.

[1] https://www.elastic.co/de/

3.5 Optical Character Recognition

Mit der optischen Zeichenerkennung (OCR) ist es möglich, analogen Text in maschinen-lesbaren digitalen Text umzuwandeln. Ob die analoge Vorlage handgeschrieben oder gedruckt ist, spielt dabei grundsätzlich keine Rolle. Die Umwandlung besteht im Normalfall aus mehreren Schritten, welche nacheinander ausgeführt werden. Begonnen wird mit einem digitalen Bild oder Scans des Text, welches dann für die nachfolgenden Schritte optimiert wird, indem beispielsweise Rauschen unterdrückt oder Farben invertiert werden. Anschließend werden die Bereiche der Vorlage, welche Text enthalten lokalisiert sowie die einzelnen Zeichen segmentiert. Die Eigenschaften eines einzelnen Zeichens werden mit einer Sammlung aus bekannten Zeichen verglichen, um naheliegendste Zeichen zu finden. Schlussendlich werden die segmentierten Zeichen wieder zusammengefügt und die entstehenden Wörter optional mit verschiedenen Techniken, beispielsweise der Grammatiküberprüfung, nachbearbeitet und als digitaler Text ausgegeben.

Typische OCR-Fehler entstehen aus der optischen Ähnlichkeit von Zeichen, wie beispielsweise dem kleinen s und der Ziffer 5. Die Fehlerrate eines mittels OCR eingelesenen Texts hängt vorrangig von der Qualität der analogen Vorlage in Bezug auf beispielsweise die Schriftart oder die Sprache des Texts ab.

3.6 OCR Error Correction

Die Autoren der Arbeit *OCR Error Correction Using Character Correction and Feature-Based Word Classification* [4] beschäftigen sich mit dem nachgelegten Korrigieren von OCR-Fehlern. Ihre Herangehensweise besteht aus drei Schritten. Für ein einzelnes Wort werden zu Beginn alle syntaktisch ähnlichen Wörter anhand des Wortschatzes der jeweiligen Sprache als Kandidaten erschlossen und anhand der Zeichenunterschiede eine Konfusionsmatrix erstellt. Anschließend wird für jeden der möglichen Kandidaten gezählt, wie oft dieser als Unigram, als Rückwärtsbigramm (also der Kandidat zusammen mit dem vorgestellten Wort im Text) und als Vorwärtsbigramm auftaucht. Aus den entstehenden Vektoren der Kandidaten wird nun anhand der genannten Eigenschaften der Kandidat mit der größten Wahrscheinlichkeit bestimmt. Schlussendlich wird der wahrscheinlichste Kandidat mit dem originalen Wort im Text verglichen und anhand aller Werte eine Entscheidung getroffen, ob das originale Wort behalten oder ersetzt wird. Im dazugehörigen Experiment konnten die Autoren eine Verringerung der Wortfehlerrate von 30% auf 21% verzeichnen, was einer Verbesserung von ca. 30% entspricht.

Kapitel 4

Eigener Ansatz

Die grundlegende Idee, SBERT resistenter gegenüber OCR-fehlerbehafteten Daten zu machen, bezieht sich auf das Training des SBERT-Models. Die Ausgangslage hierfür ist, dass es möglich ist, dem Model zeigen zu können, welche OCR-Fehler existieren und wie mit ihnen umzugehen ist. So sollte das Model bei der späteren Verwendung in der Lage sein, OCR-Fehler im übertragenen Sinne als solche zu erkennen und mit dem Text umzugehen, als wäre dieser frei von solchen Fehlern. Die Python-Bibliothek nlpaug[1] bietet verschiedene Möglichkeiten der Text-Augmentierung für Experimente im Bereich des Maschinellen Lernen. Für diese Arbeit wird der *ocraug* genutzt, mit welchem sich zufällige OCR-Fehler in definierbarer Menge simulieren lassen. Ocraug stützt sich dabei auf eine Konfusionsmatrix, welche Zeichen und Ihre wahrscheinlichsten Verwechslungskandidaten enthält. Zur Augmentierung wird ein Text eingelesen und einzelne Zeichen verändert. Mit den Parametern *aug_char* und *aug_word* lässt sich einstellen, wie viel Prozent der Zeichen eines Tokens und wie viel Prozent der Wörter des Textes augmentiert werden.

Zeichen	Augmentierung	Beispielsatz	Augmentierung
0	o , O , D	Dort ist der Ofen	D0rt ist der Dfen
b	6	Du bist schön	Du 6ist schön
B	8	Berlin	8erlin

Tabelle 4.1: OCR-Fehler Beispiele, Verwechslungen treten typischerweise bei optisch ähnlichen Zeichen auf

Für das eigentliche Training des Modells wird, den Empfehlungen der AutorInnen des SBERT Papers[2] entsprechend, die Programmiersprache Python

[1] https://github.com/makcedward/nlpaug
[2] https://www.sbert.net/

in der Programmierumgebung PyCharm genutzt. Wichtige Funktionen werden durch die bekannte Bibliothek PyTorch[3] gestellt. Beim Training eines SBERT-Models wird mithilfe eines *word_ embedding_ models* sowie einem *pooling_ model* eine Satzeinbettung einzelner Sätze in Form eines 768-dimensionalen Vektors erstellt. Anschließend werden die Ähnlichkeitswerte im Intervall [0,1] für definierte Vektor- bzw. Satzpaare übergeben und diese anhand des Wertes im Raum platziert. Ausgehend von dieser räumlichen Beziehung der Sätze ist das Model nach abgeschlossenem Training in der Lage, die Platzierung im Raum von weiteren, bis dato unbekannten Sätzen festzulegen und schlussendlich über die Distanz dieser Satzeinbettungen zueinander die Ähnlichkeitswerte für alle Sätze zu approximieren.

		Ähnlichkeit			
	Beispielsätze	A	B	C	D
A	Morgens trinke ich gerne Kaffee	1.00	0.93	0.06	0.05
B	Nach dem Aufstehen brauche ich Espresso	0.93	1.00	0.10	0.08
C	Ich begrüße das geplante Tempolimit	0.06	0.10	1.00	0.45
D	Ich fahre gerne etwas gemütlicher	0.05	0.08	0.45	1.00

Tabelle 4.2: Beispiele für die semantische Ähnlichkeit von Sätzen, Werte zwischen 0 für sehr unterschiedlich und 1 für sehr ähnlich

In dem in der Tabelle 4.2 veranschaulichten Beispiel würden die Sätze A und B also sehr nah beieinander, allerdings weit entfernt von den Sätzen C und D platziert werden. Satz C und D würden mit einer mittleren Distanz zueinander stehen, da zumindest eine gewisse thematische Verwandtschaft erkennbar ist. Außerdem besitzen alle Sätze einen Ähnlichkeitswert von 1 zu sich selbst. Als Grundlage für das Training wird der *STS Benchmark*[4] Datensatz genutzt, welcher aus rund 5.700 Satzpaaren und ihren jeweiligen Ähnlichkeitswerten besteht. Dieser ist unter anderem für Experimente zur semantischen Ähnlichkeit von Texten konzipiert und wird regelmäßig als Standard für Benchmarking verwendet [1]. Ähnlich wie in der Arbeit zu Augmented SBERT [10] vorgestellt, wird durch Kopien der Satzpaare des Datensatzes ein Silber Datensatz erstellt, welcher nach der Augmentierung mit OCR Fehlern mit dem Original Datensatz vereint wird. Der entstandene erweiterte Datensatz wird randomisiert, im üblichen Verhältnis von vier zu eins, in Trainingsdaten für das Modell und Testdaten für spätere Experimente geteilt. Bei der Augmentierung des Silber Datensatzes werden drei Herangehensweisen betrachtet.

[3] https://pytorch.org/

[4] http://ixa2.si.ehu.eus/stswiki/index.php/STSbenchmark

AA-Satzpaare Ein Satz weist logischerweise eine Nulldistanz zu sich selbst auf; das Satzpaar (A,A) besitzt also einen Ähnlichkeitswert von 1. Davon ausgehend kann von einem existierenden Satz A eine Kopie A' mit augmentierten OCR-Fehlern erstellt werden, um den Datensatz mit dem entstandenen Satzpaar (A,A') zu erweitern. Wird nun dem neuen Satzpaar (A,A') der Ähnlichkeitswert 1 zugewiesen, sollte die räumliche Platzierung durch das Modell der Sätze A und A' identisch oder zumindest nah beieinander sein.

AB-Satzpaare Bei einer weiteren Möglichkeit, die Trainingsdaten anzupassen, werden existierende Satzpaare und ihre dazugehörigen Ähnlichkeitswerte kopiert. Jedes der kopierten Satzpaare wird jeweils entweder im Satz A, im Satz B oder in beiden Sätzen mit OCR-Fehlern augmentiert und erhält denselben originalen Ähnlichkeitswert.

Definierte Fehler Letztendlich wird noch einen weiterer augmentierter Datensatz erstellt, mithilfe dessen überprüft werden soll, ob es möglich ist ein Modell auf einen einzelnen bestimmten Fehler zu trainieren. Hierbei werden für ein Satzpaar AB alle Vorkommen eines einzelnen Zeichens durch ein anderes Zeichen ersetzt, mit welchem eine OCR Technik es üblicherweise verwechseln könnte; beispielsweise die Buchstaben s und die Ziffer 5. Die dabei entstehenden Satzpaar-Kopien erhalten wie bei den AB-Satzpaaren den Original-Ähnlichkeitswert.

Satz 1	Satz 2	Ähnlichkeit
A plane is taking off.	An air plane is taking off.	1.00
A man is playing a large flute.	A man is playing a flute.	0.76

Tabelle 4.3: Beispiele für Satzpaare im STS Benchmark Datensatz

Satzpaar	Satz 1	Satz 2	Ähnlichkeit
AB	Du bist schön	Du bist cool	0.84
AA'	Du bist schön	Du 6i5t 5chon	1.00
BB'	Du bist cool	Du bisf c001	1.00
A'B	Du 6i5t 5chon	Du bist cool	0.84
AB'	Du bist schön	Du bisf c001	0.84
A'B'	Du 6i5t 5chon	Du bisf c001	0.84

Tabelle 4.4: Beispiele für augmentierte Satzpaare

Kapitel 5

Experimente

Die Experimente, mit welchen der Effekt des jeweiligen Trainings auf Resistenz des Modells gegen OCR-Fehler getestet wird, sind in zwei verschiedene Abschnitte geteilt: die Kosinus-Ähnlichkeit und die Textklassifikation. Bei beiden Herangehensweisen, welche nachfolgend noch detaillierter erläutert werden, werden vier unterschiedlich trainierte Modelle getestet:

Modell Vanilla
: trainiert ohne OCR-Fehler

Modell AA
: trainiert mit zufälligen OCR-Fehlern
nur AA-Satzpaare

Modell AB
: trainiert mit zufälligen OCR-Fehlern
nur AB-Satzpaare

Modell ABAA
: trainiert mit zufälligen OCR-Fehlern
AA- und AB-Satzpaare

Modell s5
: trainiert mit definiertem OCR-Fehler
jedes *s* ersetzt durch *5*

Alle Modelle werden mit den selben Parametern trainiert. Als Grundlage dient das WordEmbeddingModel *bert-base-uncased*[1] mit einer maximalen Sequenzlänge von 256. Trainiert wird für 5 Epochen mit 500 Aufwärmschritten,

[1] https://huggingface.co/bert-base-uncased

als Optimierer wird PyTorchs *AdamW* Algorithmus genutzt. Bei den Modellen, welche auf zufälligen OCR-Fehlern trainieren, werden einzelne Satzpaare mehrfach augmentiert, um die Datenmenge zu erhöhen. Ein Teil des Datensatzes von Model AB könnte also wie in Tabelle 5.1 aussehen.

	Satz A	Satz B	Ähnlichkeit
AB	Du bist schön	Du bist cool	0.84
A'B	Ou blst schön	Du bist cool	0.84
A'B	Du bi5t sohön	Du bist cool	0.84
AB'	Du bist schön	Du 6ist oool	0.84
AB'	Du bist schön	Du bi5t c00i	0.84
A'B'	Do bisf 5chön	Du blst cOol	0.84
A'B'	Du bist scb0n	0u bisf cool	0.84

Tabelle 5.1: Beispiel für mehrfach augmentierte Satzpaare als Trainigsdaten für Models

5.1 Kosinus-Ähnlichkeit

Unter anderem mithilfe der Kosinus-Ähnlichkeit lassen sich unterschiedlich trainierte Modelle recht simpel auf ihre Performance überprüfen. Hierbei wird der Kosinus des Winkels zwischen zwei Vektoren bestimmt. Der ermittelte Wert zwischen 0 und 1 dient als Maß für die Ähnlichkeit der Ausrichtung beider Vektoren.

$$Kosinus - Aehnlichkeit = cos(\theta) = \frac{a * b}{||a|| * ||b||} = \frac{\sum_{i=1}^{n} a_i * b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2} * \sqrt{\sum_{i=1}^{n}(b_i)^2}}$$

Für einen beim Training des Modells zurückgehaltenen Teil des Datensatzes werden die jeweiligen Werte für die Kosinus-Ähnlichkeit errechnet. Somit erhalten wir Satzpaare, für welche sowohl die Original Label für die Semantische Ähnlichkeit als auch unsere ermittelten Werte für die Kosinus-Ähnlichkeit der jeweiligen Satzeinbettungen vorliegen. Diese Werte lassen sich nun mit der Pearson Korrelation auf eine positive Korrelation überprüfen. Je höher die Korrelation zwischen Original Label und Kosinus-Ähnlichkeit ist, desto besser arbeitet das Modell in Bezug auf OCR-Fehler.

5.1.1 Datensätze

Die Wahl eines Testdatensatzes gestaltet sich bei dieser Herangehensweise simpel. Der Testanteil des in Kapitel 4 genannten *STS Benchmark* Datensatzes

eignet sich für das Vergleichen von Ähnlichkeitswerten bei Satzpaaren. Weiterführend werden noch zwei Kopien des Datensatzes erstellt und wie folgt augmmentiert:

Datensatz Original
Originaler Zustand ohne OCR-Fehler

Datensatz Zufallsfehler
Augmentiert mit zufälligen OCR-Fehlern
alle möglichen Kombinationen (siehe Tabelle 5.2)

Datensatz Definiert
Augmentiert mit definiertem OCR-Fehler | *s* ersetzt durch *5*
alle möglichen Kombinationen (siehe Tabelle 5.2)

Kombination	Satz A	Satz B
AB	Original	Original
A'B	Augmentiert	Original
AB'	Original	Augmentiert
A'B'	Augmentiert	Augmentiert

Tabelle 5.2: Satzpaar-Kombinationen zum Testen der verschiedenen Modelle in Bezug auf die Kosinusdistanz | Kombinationen werden mehrmals augmentiert, um die Datenmenge zu erhöhen (siehe 5.1)

Ergebnisse

Anhand der Testergebnisse des Originaldatensatzes in Tabelle 5.3 ist deutlich zu erkennen, dass Modell AA und ABAA, welche unter anderem auf mit OCR-Fehlern augmentierten AA-Satzpaaren trainiert wurde, nahezu unbrauchbare Ergebnisse liefern. Modell AB und s5 schneiden jedoch gleich gut wie das Modell Vanilla ab. Beim Testen der mit OCR-Fehlern augmentierten Datensätze können wir eine immense Steigerung der *Kosinusdistanz* bei Modell AB und s5 feststellen. Die größte Verbesserung finden wir beim Test zum Datensatz Zufallsfehler in Form einer Verdopplung des vom Evaluator zurückgegebenen Wertes, wenn wir die Ergebnisse von Modell Vanilla und Modell AB vergleichen. Interessant ist außerdem, dass Modell s5 ein weitestgehend ähnlich gutes Ergebnis liefert und das Modell AB bei Datensatz Definiert sogar übertrifft. Das Training auf augmentierten AA-Satzpaaren wirkt sich insgesamt negativ auf die Ergebnisse der jeweiligen Modelle aus. So liefern die Modelle AA und ABAA die schlechtesten Werte, bei Datensatz Original und Definiert sogar stark unter dem Wert des Vanilla Modells.

Datensatz	Vanilla	Model AA	Model AB	Model ABAA	Model s5
Original	0.79	0.21	0.78	0.31	**0.80**
Zufallsfehler	0.22	0.18	**0.44**	0.26	0.39
Definiert	0.53	0.19	0.68	0.29	**0.72**

Tabelle 5.3: Ergebnisse der Kosinusdistanz beim Testen aller genannten Models und Datensätze als Spearmankorrelationswert | beste Werte hervorgehoben

5.2 Textklassifizierung

Die Textklassifizierung bildet einen wichtigen Aufgabenbereich der Natürlichen Sprachverarbeitung in Bezug auf unterschiedliche Anwendungen wie beispielsweise der Erkennung von Spam oder der Kategorisierung von Rezensionen oder Fehlerberichten. Dabei soll ein Klassifizierungsmodell oder ein spezieller Algorithmus einen Text einer Klasse aus einer zuvor festgelegten Klassenmenge zuweisen können. Für die Textklassifizierung wird ein Klassifikator in Form einer *SupportVectorMachine* für das Kategorisieren von Text trainiert. Hierbei lernt der Klassifikator beispielsweise, ob ein Satz ein positives oder negatives Sentiment besitzt (siehe 5.4). Anschließend wird auf einem zurückgehaltenen Teil des Datensatzes getestet, wie gut der Klassifikator die zuvor genannten Kategorien erkennen kann. An dieser Stelle ist zu verdeutlichen, dass das Training der SBERT-Modelle bereits abgeschlossen ist. Das Trainieren der Klassifikatoren hat keinen Einfluss auf die SBERT-Modelle selbst.

Satz	Sentiment
Du bist schön	Positiv
Dieser Saft schmeckt großartig	Positiv
Der Film war langweilig	Negativ

Tabelle 5.4: Beispiele für das Sentiment eines Satzes

5.2.1 Textklassifizierung *IMDB*

Für die Textklassifizierung bieten sich viele öffentlich zugängliche Datensätze an. Für den ersten Klassifizierungstestwird der *imdb*-Datensatz[2] genutzt, welcher 50.000 englischsprachige Filmrezensionen beinhaltet, die mit dem Label 0 für *negativ* und dem Label 1 für *positiv* gekennzeichnet sind. Durchschnittlich

[2]https://huggingface.co/datasets/imdb

enthält eine Zeile des Datensatzes, also eine Rezension, 200 Wörter. Für unser Experiment wird der Datensatz randomisiert auf 20.000 Rezensionen begrenzt, im Verhältnis vier zu eins in Trainings- und Testdaten aufgeteilt und vier Kopien der resultierenden Datenmenge erstellt. Anschließend werden die Daten wie folgt augmentiert:

Szenario Fehlerfrei
Trainingsdaten ohne OCR-Fehler
Testdaten ohne OCR-Fehler

Szenario LowLow
Trainingsdaten mit wenig OCR-Fehlern
Testdaten mit wenig OCR-Fehlern

Szenario HighHigh
Trainingsdaten mit vielen OCR-Fehlern
Testdaten mit vielen OCR-Fehlern

Szenario TestHigh
Trainingsdaten ohne OCR-Fehler
Testdaten mit vielen OCR-Fehlern

Szenario TrainHigh
Trainingsdaten mit vielen OCR-Fehlern
Testdaten ohne OCR-Fehler

Ergebnisse *IMDB*

	Modell				
Szenario	Vanilla	AA	AB	ABAA	s5
Fehlerfrei	**0.86**	0.77	**0.86**	0.81	**0.86**
LowLow	**0.82**	0.73	**0.82**	0.79	**0.82**
HighHigh	**0.72**	0.63	0.68	0.66	0.69
TestHigh	0.57	0.57	0.61	**0.62**	0.60
TrainHigh	0.63	0.69	**0.79**	0.74	**0.79**

Tabelle 5.5: Datensatz imdb | Ergebnisse der Textklassifikation beim Testen aller genannten Models und Datensätze als F1-Score (average) | beste Werte hervorgehoben

Grundsätzlich können die Ergebnisse der ersten Textklassifizierung in Tabelle 5.5 die Ergebnisse der Kosinusdistanz nicht bestätigen. Bei den Szenarien Fehlerfrei, LowLow und HighHigh liefert das Modell Vanilla unter anderem die besten Werte. Beim Szenario TestHigh lassen sich keine nennenwerten Verbesserungen oder Verschlechterungen erkennen. Einzig bei Szenario TrainHigh performen die Modelle AB und s5 circa 25 Prozent besser. Hierbei muss allerdings erwähnt werden, dass dieses Szenario in der Praxis kaum vorkommen sollte. Auch bei der Textklassifikation liefern die Modelle, welche auf AA-Satzpaaren trainiert wurden, insgesamt die schlechtesten Ergebnisse. So schneiden die Modelle AB und ABAA bei den ersten drei Szenarien am schlechtesten ab.

5.2.2 Textklassifizierung *Hatespeech*

Der Datensatz *tweets_hate_speech_detection*[3] wird für den zweiten Textklassifizierungstest genutzt. Dieser besteht aus ca. 50.000 englischsprachigen Tweets, welche durchschnittlich 13 Wörter enthalten und mit dem Label 0 für *No Hatespeech* und dem Label 1 für *Hatespeech* gekennzeichnet sind. Für das Experiment wird ein Teil des Datensatzes mit 20.000 randomisiert ausgewählten Zeilen bzw. Tweets genutzt, welcher im Verhältnis Vier zu Eins in Trainings- und Testdaten aufgeteilt wird. Die Augmentierung der OCR-Fehler gleicht den Szenarien der ersten Textklassifizierung mit dem *imdb*-Datensatz.

Ergebnisse *Hatespeech*

	Modell				
Szenario	Vanilla	AA	AB	ABAA	s5
Fehlerfrei	0.78	0.71	0.76	0.72	**0.79**
LowLow	0.76	0.70	0.75	0.71	**0.77**
HighHigh	**0.73**	0.66	0.71	0.68	**0.73**
TestHigh	0.66	0.63	0.67	0.63	**0.68**
TrainHigh	0.73	0.68	**0.74**	0.71	**0.74**

Tabelle 5.6: Datensatz tweets_hate_speech_detection | Ergebnisse der Textklassifikation beim Testen aller genannten Modelle und Datensätze als F1-Score (average) | beste Werte hervorgehoben

Die Ergebnisse der Textklassifizierung anhand des Hatespeech-Datensatzes in Tabelle 5.6 bestätigen erneut, dass das Training der Modelle AA, AB und

[3]https://huggingface.co/datasets/tweets_hate_speech_detection

ABAA keinen positiven Effekt auf die Performanz bezüglich OCR-Fehlern hat - das Modell Vanilla liefert fast ausschließlich bessere Werte. Das Modell s5 erzielt in allen fünf Szenarien die besten Werte.

5.2.3 Nähere Betrachtung Modell s5

Obwohl die bisherigen Ergebnisse der verschiedenen Experimente keinen eindeutigen positiven Effekt erkennen lassen, erwecken die Werte des Modells s5 den Eindruck, dass die angewandte Herangehensweise funktionieren kann. Daraus folgernd wird abschließend ein detaillierterer Blick auf die Performanz der Modelle Vanilla und s5 in Bezug auf den definierten OCR-Fehler (*s ersetzt 5*) geworfen. Für diesen Vergleich wird erneut der IMDB-Datensatz verwendet und mit dem definierten OCR-Fehler für folgende Szenarien versehen:

Szenario None_s5
Trainigsdaten ohne OCR-Fehler
Testdaten mit definiertem OCR-Fehler

Szenario s5_s5
Trainigsdaten mit definiertem OCR-Fehler
Testdaten mit definiertem OCR-Fehler

Ergebnisse

Szenario	Modell Vanilla	Modell s5
None_s5	0.78	**0.82**
s5_s5	0.83	**0.84**

Tabelle 5.7: Datensatz imdb | Ergebnisse der Textklassifikation beim Testen der Modelle Vanilla und s5 als F1-Score (average) | beste Werte hervorgehoben

Die Ergebnisse in Tabelle 5.7 bestätigen den beobachteten Effekt, dass sich das Modell s5 resistenter gegen die definierten s5-Fehler verhält. Die resultierende Verbesserung des F1-Scores von durchschnittlich 3% ist zwar gering, allerdings liefert das Modell Vanilla ohnehin schon recht gute Ergebnisse - eine größere Verbesserung wäre also unrealistisch.

Kapitel 6

Fazit

Ziel dieser Arbeit ist es, herauszufinden ob es möglich ist, SBERT-Modelle auf eine bestimmte Art zu trainieren, welche die Performanz der Modelle bei der Verarbeitung von Daten mit OCR Fehlern steigert. Dafür wurden fünf SBERT-Modelle auf insgesamt 68.000 einzigartigen Satzpaaren trainiert, wovon 64.000 Satzpaare mit OCR Fehlern augmentiert waren. Die Modelle wurden sowohl mittels Kosinus-Ähnlichkeit als auch anhand als Aufgabe der Natürlichen Sprachverarbeitung üblichen Textklassifizierungen getestet.

Die Kosinus-Ähnlichkeit wurde mittels 10.000 einzigartiger Satzpaaren getestet, von denen 9.000 augmentiert waren. Beim **Datensatz Definiert** lieferte das **Modell s5** mit einer Verbesserung von 36% im Vergleich zum **Modell Vanilla** das beste Ergebnis, dicht gefolgt von **Modell AB** (+28%). Das **Modell AB** konnte dafür beim Test mit **Datensatz Zufallsfehler** eine Verbesserung von 100% aufweisen. Die Ergebnisse der **Modelle AB** und **s5** sprechen dafür, dass der Ansatz Erfolg haben kann. Die **Modelle AA** und **ABAA** lieferten die schlechtesten Ergebnisse. Daraus ergibt sich, dass der Ansatz der AA-Satzpaare (siehe Kapitel 4) zumindest in angewendeten Form nicht funktioniert. Wieso die Modelle, welche unter anderem auf AA-Satzpaaren trainiert wurden, vergleichsweise so schlecht abschneiden, kann nicht abschließend beurteilt werden. In Bezug auf den Test zur Kosinus-Ähnlichkeit kann die Hypothese aufgestellt werden, dass das Trainieren eines SBERT-Modells auf OCR Fehlern dessen Resistenz gegen OCR Fehler steigern kann.

In weiteren Experimenten wurden drei verschiedene Textklassifizierungen mit den Modellen durchgeführt. Für die Klassifizierung des *IMDB*-Datensatzes wurden insgesamt 48.000 Trainings- und 12.000 einzigartige Testzeilen verwendet, von welchen zwei Drittel mit künstlichen OCR Fehlern versehen waren. Die Ergebnisse dieses Experiments sind uneindeutig, wobei das **Modell Vanilla** am besten abzuschneiden scheint. Die **Modelle AB** und **s5** erreichen ähnliche Werte wie das **Modell Vanilla**, können aber keine Besserung auf-

weisen. Lediglich beim **Szenario TrainHigh** erreichen Sie einen rund 25% besseren Wert, jedoch ist dieses Szenario recht praxisfern. Auch hier schneiden die **Modelle AA** und **ABAA** erneut am schlechtesten ab, wobei das **Modell ABAA** mit dem besten Ergebnis im **Szenario TestHigh** einen kleinen Ausreißer macht.

Bei der nächsten Klassifizierung, bei welcher Tweets dahingehend klassifiziert werden, ob sie Hassrede enthalten oder nicht, wurden die fünf Modelle auf insgesamt 60.000 Tweets trainiert und getestet, von welchen 40.000 mit OCR Fehlern augmentiert waren. Erneut lieferten die **Modelle AA** und **ABAA** durchweg die schlechtesten Ergebnisse. Die **Modelle Vanilla**, **AB** und **s5** schneiden sehr ähnlich ab, wobei das **Modell s5** bei diesem Experiment in allen fünf Szenarien Bestwerte liefert. Die Hypothese aus dem Experiment zur Kosinus-Ähnlichkeit verfestigt sich damit.

Um den Effekt des Trainings anhand eines einzelnen, definierten OCR Fehlers genauer zu überprüfen, werden in einer letzten Klassifizierung das Ergebnis des **Modells Vanilla** mit dem des **Modells s5** verglichen. Als Trainings- und Testdaten dienen erneut die Filmrezensionen des *IMDB*-Datensatzes, welche diesmal nicht mit zufälligen, sondern dem definierten OCR Fehler s5 versehen wurden. Bei beiden getesteten Szenarien erreichte das **Modell s5** den besseren Wert.

Zusammenfassend kommen wir zu der Hypothese, dass es möglich ist, die Performanz eines SBERT-Modells bei der Verarbeitung von Daten mit OCR Fehlern durch das Training des Modells zu steigern. In weiteren Arbeiten gilt es herauszufinden, weshalb AA-Satzpaare einen eher negativen Einfluss auf die Performanz der Modelle zu haben scheinen. Außerdem sollte der Ansatz des definierten OCR Fehlers weiter geprüft werden, indem man die Datensätze für das Training sowie die Experimente beispielsweise nach und nach mit weiteren, definierten OCR Fehlern augmentiert.

Literaturverzeichnis

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SyK00v5xx`.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL `https://doi.org/10.18653/v1/n19-1423`.

[3] Rose Holley. How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs. *D Lib Mag.*, 15 (3/4), 2009. doi: 10.1045/march2009-holley. URL `https://doi.org/10.1045/march2009-holley`.

[4] Ido Kissos and Nachum Dershowitz. OCR error correction using character correction and feature-based word classification. *CoRR*, abs/1604.06225, 2016. URL `http://arxiv.org/abs/1604.06225`.

[5] Daniel P. Lopresti. Optical character recognition errors and their effects on natural language processing. *Int. J. Document Anal. Recognit.*, 12(3): 141–151, 2009. doi: 10.1007/s10032-009-0094-8. URL `https://doi.org/10.1007/s10032-009-0094-8`.

[6] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and

Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

[7] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

[8] Ilona Regulski. The Origins and Early Development of Writing in Egypt. In *The Oxford Handbook of Topics in Archaeology*. Oxford University Press, 2016. URL https://doi.org/10.1093/oxfordhb/9780199935413.013.61.

[9] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

[10] Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. Augmented SBERT: data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. *CoRR*, abs/2010.08240, 2020. URL https://arxiv.org/abs/2010.08240.