Argument Mining

Modul Projekt

Sandra Sánchez

WiSe 23/24

Postdam Universität

Der vorliegende Bericht dokumentiert die Experimente, die mit dem berühmten Korpus von Stab und Gurevych (2017) durchgeführt wurden, das von Schäfer und Stede (2023) erweitert wurde und das wir als Referenz verwenden. Wir haben die ursprünglichen Annotationen und alle Texte abgerufen und daraus eine Pipeline aufgebaut, in der wir die Vorverarbeitung, die Feature-Extraktion, das Modelltraining und die Klassifizierung sowie die anschließende Evaluierung durchgeführt haben. Ziel ist es, die Leistung und Genauigkeit bei der Identifizierung argumentativer Komponenten eines einfachen ML-Klassifikators wie dem RandomForestClassifier mit dem hochentwickelten vortrainierten Roberta-Modell zu vergleichen, das von Schäfer und Stede vorgeschlagen wurde. Die Frage, die wir beantworten wollen, ist, was passiert, wenn wir einem einfachen RFC-Modell eine Merkmalsextraktion hinzufügen, und wie genau sind diese Vorhersagen im Vergleich zu denen von RoBERTa.

Wie in der Arbeit von Schäfer und Stede besteht unser Ziel darin, Flussmuster in den verschiedenen Argumentationskomponenten zu identifizieren, die es uns ermöglichen, Behauptungen und Prämissen und ihre jeweiligen Subtypen korrekt zu lokalisieren.

Towards Fine-Grained Argumentation Strategy Analysis in Persuasive Essays

Der Bereich des Argument Mining konzentriert sich auf das Aufspüren argumentativer Elemente in natürlicher Sprache und die Beziehungen zwischen ihnen. Schäfer und Stede interessieren sich in ihrer Arbeit für die sprachlichen Strategien der Persuasion, was mit der Auswahl der richtigen Argumente zu tun hat, aber auch damit, wie diese angeordnet sind. Um diese Flussmuster zu identifizieren, verwenden sie ihre eigenen Annotationen, die eine Verbesserung derjenigen von Stab und Gurevych darstellen sollen.

In diesem Projekt werden wir ihren Ansatz nicht reproduzieren, aber wir werden denselben Korpus und dieselben Annotationen sowie das für die Klassifizierungsaufgabe vorgeschlagene Transformersmodell verwenden.

Formatierung und Preprocessing des Korpus

Der Korpus besteht aus 402 Essays. Der von Schäfer und Stede vorgeschlagene Etikettensatz[1] besteht aus den folgenden Claims: *policy*, *value* und *fact*; und den Prämissen *testimony*, *statistics*, *hypothetical instance*, *real-example* und *common-ground*.

Da sich die Annotationen auf Satzfragmente (Nebensätze, von denen auch Konnektoren ausgeschlossen sind) und nicht auf ganze Sätze beziehen, haben wir uns entschieden, die Texte in der ersten Phase nach diesen Annotationen zu organisieren. Zunächst haben wir alle annotierten Texte ausfindig gemacht und in einer kontextrelevanten Reihenfolge in der Datei all_sorted_annotated_texts.txt angeordnet, in der die Artikelnummer, die Art der argumentativen Komponente und der Text, auf den sich bezogen wird, angegeben sind. Von dort aus generiert get_labelled_sentences_from_data die Rohtexte zusammen mit den entsprechenden Tags. Wir identifizieren die Fragmente, denen ein Tag-Typ entspricht, und weisen ihn ihnen zu. Alle anderen erhalten das Tag *non-argumentative*. Auf diese Weise hoffen wir, Muster leichter erkennen zu können.

Die Vorverarbeitung der einzelnen Textfragmente besteht aus der Tokenisierung, der Entfernung von Satzzeichen, der Filterung von Stoppwörtern und der Lemmatisierung. Da nach diesen Schritten viele leere Elemente vorhanden waren, haben wir auch diese Filterung durchgeführt. Dies sind die Schritte, die für das Training unserer Baseline, dem RandomForestClassifier, durchgeführt wurden, da,

[1] https://github.com/discourse-lab/arg-essays-semantic-types

wie wir sehen werden, das mit RoBERTa trainierte Modell seinen eigenen Tokenisierer verwendet.

Da wir mit Multilabel- oder besser Multiklassenklassifikation arbeiten, mussten wir schließlich die Labels kodieren, die zusammen mit dem bereits verarbeiteten Text das richtige Format haben, um vom Modell verarbeitet zu werden.

Wir müssen jedoch darauf hinweisen, dass wir aufgrund der Verwechslung von Multilabel und Multiklasse anfangs den falschen Kodierer verwendet haben, was wir durch die Beobachtung von mehr Instanzen im Support als vorhanden erkannt haben (und was wir durch das Debuggen jeder verwendeten Funktion vor der Erstellung des Klassifikationsberichts überprüft haben). Durch die Verwendung des korrekten Kodierers und somit die Zuweisung eines einzigen Labels für jeden Text wurden unsere Ergebnisse ebenfalls verbessert, und zwar von einer Genauigkeit von etwa 0.30 auf über 0.50 im Devset.

Feature Extraction

Die Merkmalsextraktion erhöht potenziell den Wert unseres Modells, indem sie die intrinsischen Fähigkeiten der ML-Architektur ergänzt, und sollte der grundlegende Schritt sein, der die Erkennung bestimmter Muster oder Überredungsstrategien garantiert. Zu diesem Zweck haben wir über die Merkmale nachgedacht, die direkt mit dem Vorhandensein argumentativer Komponenten in Verbindung gebracht werden können, und schließlich beschlossen, uns auf Folgendes zu konzentrieren: Ngrams, Topic Modelling, Dependency Relations (für die wir spacy[2] verwendet haben) und die Identifizierung von Verben in Verbindung mit Claims.

Die größte Herausforderung in dieser Phase bestand darin, die Relevanz und den Informationsgehalt der einzelnen Merkmale zu bestimmen. Zu diesem Zweck experimentierten wir mit der Art der verwendeten Features, ihrer Anzahl (ob wir z. B. zwei oder drei extrahierten) und schließlich ihrem Vorhandensein oder Fehlen. Wie

[2] https://spacy.io/usage

wir weiter unten sehen werden, entsprachen die Ergebnisse bzw. die Auswirkungen auf die Leistung letztendlich nicht den Erwartungen.

Training und Klassifizierung

Training und Klassifizierung
Sobald die Daten verarbeitet sind, gehen wir zur Trainingsphase des Modells über. Hierfür haben wir zwei Möglichkeiten untersucht: ein hochmodernes transformatorbasiertes Modell, wie RoBERTa, und einen traditionellen Klassifikator für maschinelles Lernen wie Random Forest.

Es ist erwähnenswert, dass wir die Art und Weise der Vorverarbeitung ändern mussten, da RoBERTa seinen eigenen Tokenisierer verwendet und wir die Daten anfangs tokenisiert haben, als wir Texte und ihre entsprechenden Labels gruppiert haben. Nach der Modifikation generierten wir den kompletten Satz von Texten und Tags, teilten ihn in Trainings-, Dev- und Test-Splits auf und speicherten diese Splits im Pickle-Format für die spätere Verwendung und um sicherzustellen, dass beide Klassifikatoren mit denselben Daten arbeiten.

Random Forest Classifier

Um unser Modell zu trainieren, mussten wir die Daten zunächst in ein geeignetes Format umwandeln. Die Funktion create_feature_matrix in classify_rfc.py erstellt zu diesem Zweck eine Matrix. Wir erstellen eine Instanz von TfidfVectorizer(), in die die vorverarbeiteten Texte eingebettet werden. Die Bezeichnungen werden in Numpy-Arrays umgewandelt. Diese Funktion ist dazu gedacht, optional Merkmale zu extrahieren. Falls wir diese berechnen wollen, werden wir dies für die vier oben genannten Funktionen tun. Die resultierenden Merkmale werden in ein kompatibles Format konvertiert und dann mit dem bereits existierenden Array verkettet. Die resultierenden Matrizen für Train-, Dev- und Test-Sets werden standardmäßig in Pickle-Dateien gespeichert und können direkt geladen werden, so dass wir bei jeder

nachfolgenden Ausführung des Programms Speicherplatz sparen, aber dieser Schritt kann auch übersprungen werden.

Anschließend wird eine Instanz von RandomForestClassifier erzeugt, an die die transformierten Daten angepasst werden. Dieser Klassifikator wird Vorhersagen über die Dev- und Test-Sets treffen. Im letzten Schritt wird die Genauigkeit der getroffenen Vorhersagen berechnet und ein vollständiger Klassifizierungsbericht über die Erkennung jedes einzelnen Labels erstellt.

Das Training dieses Modells war insofern keine Herausforderung, als wir mit dem Prozess und den Encoding-Techniken vertraut sind, obwohl, wie wir bereits angekündigt haben, der schwierigste Teil darin bestand, eine echte Leistung aus den extrahierten Merkmalen herauszuholen. Ursprünglich wollten wir auswählen können, welche Merkmale zusätzlich berechnet werden sollten, aber schließlich entschieden wir uns dafür, alle oder keines der Merkmale zu verwenden, da der Unterschied minimal war.

RoBERTa

RoBERTa, das Modell von HuggingFace[3] steht für "Robust optimierter BERT-Ansatz" und ist eine Variante des BERT-Modells (Bidirectional Encoder Representations from Transformers), das von Facebook AI entwickelt wurde. Es stellt einen bedeutenden Fortschritt in der Verarbeitung natürlicher Sprache dar. Es handelt sich um ein Modell, das mit großen Datensätzen und längeren Trainingssequenzen trainiert wurde und zu einer verbesserten Leistung bei verschiedenen NLP-Aufgaben führt.

Eine der größten Herausforderungen des Projekts war die rechenintensive Natur dieses Modells, was bedeutete, dass jede Ausführung des Skripts auf dem gesamten Korpus bis zu 11 Stunden dauerte. Im Vergleich zu RoBERTa ist RFC im Sinne der Ressourcennutzung viel effizienter, da es Vorhersagen auf der Grundlage von

[3] https://huggingface.co/docs/transformers/model_doc/roberta

Entscheidungsbäumen trifft. Obwohl roBERTa weniger schnell ist und mehr Speicherplatz benötigt, erfasst es viel bessere Nuancen und ist viel genauer in seinen Vorhersagen. Um diesen Ressourcenverbrauch zu vermeiden, arbeiten wir in der Entwicklungsphase mit einem sehr kleinen Teil des Korpus.

Ein weiterer erwähnenswerter Aspekt ist, dass RoBERTa einen eigenen Tokeniser verwendet, was sich natürlich auf die Leistung des Tokenisers auswirkt. Während der Entwicklungsphase haben wir versucht, das Modell mit den vorverarbeiteten Texten auf die gleiche Weise zu trainieren, wie wir es mit RFC getan haben, aber das Ergebnis war nicht sehr gut.

Für mich persönlich ist der neuartigste Aspekt des Projekts zweifellos die Arbeit mit Tensoren, die zu einer langen Implementierungszeit führte, da mehrere Formatierungsschritte erforderlich waren. Außerdem mussten wir die Klassen CustomDataCollator, die als data_collator für das Modell dient, und InputFeatures erstellen, die bei der Umwandlung der Daten in Tensoren hilft, mit denen das Modell arbeiten kann. Auf diese Weise erleichtern wir die Prozesse.

Herausforderungen

Neben dem bereits erwähnten hohen Ressourcenverbrauch und der Neuheit, mit Tensoren zu arbeiten, bestand eine weitere Herausforderung bei der Arbeit mit der Transformers-Architektur in der mangelnden Kompatibilität der Dependencies, insbesondere der Pakete click, typer und spacy, die wir schließlich durch die Erstellung einer speziellen virtuellen Umgebung für das Projekt lösten.

Der hohe Speicherverbrauch dieses Modells verhinderte außerdem, dass wir in der ersten Phase überhaupt ein Modell trainieren konnten, da das Skript vor Beginn des Trainings unterbrochen wurde. Da der Aktivitätsmonitor ständig geöffnet war, konnten

wir den Verbrauch der einzelnen Prozesse überwachen und die unnötigen Prozesse beenden. So konnten wir unser Modell erfolgreich trainieren.

Ein weiterer erwähnenswerter Aspekt bei diesem Modell ist, dass wir bei der Konvertierung der Tokenizer-Daten in ein anderes Format nicht die Möglichkeit hatten, die leeren Fragmente zu eliminieren (wie wir es bei RFC getan haben), was zur Folge hatte, dass wir in der endgültigen Auswertung 655 Instanzen von nicht-argumentativen gefunden haben, verglichen mit 365 in RFC, was sich zweifellos auf die Leistung des Modells auswirkt.

Auswertung

Während der Entwicklung stellten wir fest, dass die Leistung von RoBERTa der von RFC (mit oder ohne Merkmalsextraktion) weit überlegen war, so dass uns die Beschränkung auf ein grundlegendes Training des Transformatorenmodells - ohne Fine-Tuning - angemessen erschien, um unsere These zu testen.

Nachdem unsere Modelle trainiert waren, haben wir ihre Leistung bewertet und ihre Effizienz bei der Klassifizierung von Sequenzen verglichen. Um die Leistung der Modelle umfassend zu bewerten, haben wir eine Reihe von Bewertungskennzahlen verwendet, darunter accuracy, precision, recall und F1-score. Wir haben die Leistung beider Modelle auf der Dev und der Test Sets bewertet, um zu sehen, welchen Einfluss die Tatsache hat, dass die Daten von den Modellen zuvor noch nicht gesehen wurden, und haben in beiden Fällen eine bemerkenswerte Fähigkeit zur Generalisierung über ungesehene Daten abgeleitet, da die RFC-Genauigkeit 0.547 für die Development Set und 0.554 für die Test Set beträgt. Dieses Muster wiederholt sich im Fall von RoBERTa, das 0.786 für den Development Set und sogar 0.801 für den Testsatz erreicht.

Im Fall von RFC ist die Leistung des Modells ohne Merkmalsextraktion erwähnenswert (Abb. 1), die in unserem Fall zeigt, dass in dieser Phase etwas schief gelaufen ist und wir wahrscheinlich mehr Tests hätten durchführen oder andere/mehr Merkmale berechnen sollen. In jedem Fall ist die Auswirkung dieser Faktoren gering.

Features/Dataset	Dev Set	Test Set
RFC No features	0.5417	**0.5925**
RFC Feature Extraction	0.5469	0.5542
RoBERTa	**0.7867**	**0.8011**

Fig. 1 RFC Accuracy scores mit und ohne Feature Extraction bei RFC

Die Accuracy des Dev-Sets ohne Merkmalsextraktion liegt bei 0,54 und damit etwas niedriger als unser mit Merkmalsextraktion trainiertes Modell. Überraschend ist, dass die Accuracy für den Testsatz 0,59 erreicht.

	Random Forest Classifier							
	Precision		Recall		F1-Score		Support	
Class	Devset	Testset	Devset	Testset	Devset	Testset	Devset	Testset
common_ground	0.3118	0.3094	0.7584	0.7797	**0.4419**	**0.4430**	178	177
fact	0.0	0.0	0.0	0.0	0.0	0.0	42	36
hypothetical_instance	0.5263	0.6000	0.1176	0.1277	0.1923	0.2105	85	94
non-argumentative	0.8703	0.8856	0.8969	0.9123	**0.8834**	**0.8988**	359	365
other	0.0	0.0	0.0	0.0	0.0	0.0	0	1
policy	1.0	0.6000	0.0370	0.909	0.0714	0.1579	27	33
real_example	0.2500	0.5000	0.0141	0.0125	0.0267	0.0244	71	80
statistics	0.0	0.0	0.0	0.0	0.0	0.0	48	28
testimony	0.0	0.0	0.0	0.0	00.	0.0	3	1
value	0.4231	0.4153	0.3793	0.3224	**0.4000**	**0.3630**	145	152

Fig. 2 Classification Report von RFC

Der eigentliche Highlight unserer Experimente ist der Vergleich zwischen RFC und RoBERTa, wobei das Transformers-Modell ohne Fine-Tuning eine um 50% höhere Accuracy erreicht.

	RoBERTa							
	Precision		Recall		F1-Score		Support	
Class	Devset	Testset	Devset	Testset	Devset	Testset	Devset	Testset
common_ground	0.5130	0.5707	0.5562	0.6158	**0.5337**	**0.5924**	178	177
fact	0.2222	0.2500	0.0476	0.0556	0.0784	0.0909	42	36
hypothetical_instance	0.5833	0.5625	0.5765	0.5745	0.5799	0.5684	85	94
non-argumentative	0.9985	1.0	1.0	1.0	**0.9992**	**1.0**	658	655
other	0.0	0.0	0.0	0.0	0.0	0.0	0	1
policy	0.5429	0.8571	0.7037	0.7273	**0.6129**	**0.7869**	27	33
real_example	0.7692	0.7377	0.7042	0.5625	**0.7353**	**0.6383**	71	80
statistics	0.4359	0.3056	0.3542	0.3929	0.3908	0.3438	48	28
testimony	0.0	0.0	0.0	0.0	0.0	0.0	3	1
value	0.5491	0.5879	0.6552	0.7039	**0.5975**	**0.6407**	145	152

Fig. 3 Classification Report von RoBERTa

Da die Genauigkeit allein nicht viel über die Vorhersage der einzelnen Klassen aussagt, erstellen wir einen vollständigen Classification Report. Hier sehen wir den Gesamteinfluss der Support. Im Allgemeinen gilt: Je mehr Instanzen einer Klasse das Modell gesehen hat, desto besser kann es sie vorhersagen. Dies gilt für die Klasse der nicht-argumentativen Texte (d. h. alle nicht etikettierten Texte), die am zahlreichsten ist und folglich höhere Werte erhält, sowie für die *testimony* und die *other* Klassen, die einen Wert von 0,0 für 1 Instanz aufweisen. Andererseits gibt es argumentative Komponenten, die trotz ihrer Seltenheit erstaunlich gut erkannt werden: *policy* erhält einen F1 von 0,78 für 33 Instanzen und *real_exampl*e einen 0,63 für 80.

Fazit

Obwohl wir nicht dieselben Schritte befolgt und nach denselben Strukturen wie Schäfer und Stede gesucht haben, da wir keine Hierarchie von Claims und deren Beziehungen zu anderen Komponenten unterschieden, sondern jedem Textfragment

nur eines der 10 Labels zugewiesen haben (und daher unsere Daten nicht vergleichen können), können wir einen parallelen Fortschritt beobachten, da auch sie eine dramatische Steigerung der Leistung von RoBERTa im Vergleich zu einer Baseline festgestellt haben.

Wir sind zu dem Schluss gekommen, dass die Extraktion von Merkmalen eine sehr mühsame Arbeit ist und vielleicht nur Sinn macht, wenn man viele Merkmale berechnet und durch Ausprobieren die effektivsten findet.

Außerdem garantiert die Verwendung eines vorab trainierten neuronalen Modells gute Ergebnisse bei der Klassifizierung von zuvor ungesehenen Daten, und wir haben auch gesehen, dass wir durch die Verwendung dieses Modells in der Lage waren, eine hohe Genauigkeit bei den Vorhersagen zu erreichen, obwohl wir einen unzureichend ausgewogenen Datensatz hatten, so dass wir vermuten, dass die ausführliche Annotationsarbeit von Schäfer und Stede (2023) ebenfalls einen großen Einfluss hatte.

Referenzen

Discourse-lab/arg-essays-semantic-types. (2023). [Software]. Discourse Research Lab. https://github.com/discourse-lab/arg-essays-semantic-types (Original work published 2023)

RoBERTa. (o. J.). Abgerufen 6. März 2024, von https://huggingface.co/docs/transformers/model_doc/roberta

Schaefer, R., Knaebel, R., & Stede, M. (2023). Towards Fine-Grained Argumentation Strategy Analysis in Persuasive Essays. In M. Alshomary, C.-C. Chen, S. Muresan, J. Park, & J. Romberg (Hrsg.), *Proceedings of the 10th Workshop on Argument Mining* (S. 76–88). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.argmining-1.8

Stab, C., & Gurevych, I. (2017). Parsing Argumentation Structures in Persuasive Essays. *Computational Linguistics*, *43*(3), 619–659. https://doi.org/10.1162/COLI_a_00295